UNIVERSITY OF OSLO
Department of Informatics

Collecting Sensor Data for Active Music

Masters Thesis in Microelectronic Systems (60 pt)

Eirik Renton

June 2, 2009

Abstract

This thesis is about motion registration for *Active Music* applications, where motions is used to generate sound and music in real-time.

Different motion capture systems have been implemented and tested in this master thesis. These systems consist of sensor elements, which wirelessly transfers motion data to a receiver element. The sensor elements consist of a microcontroller, accelerometers and a radio transceiver. The receiver element consists of a radio receiver connected to a computer for real time sound synthesis. The wireless transmission between the sensor elements and the receiver element is based on the low rate IEEE 802.15.4/Zigbee standard. Transmission delays have to be unnoticeable for users of *Active Music* applications. Various tests considering transmission delays have been conducted in order to find advantages and disadvantages of the implemented motion capture systems. Limitations such as physical size of the systems and processing capabilities within the systems have been looked into.

The first implemented system is based on a star topology. This implementation has an upper limit of 3 sensor elements for each receiver element. A more theoretical calculation shows the possibility for 6 sensor elements in a more effective implementation. The second implemented system is based on a peer-to-peer topology, where one sensor element reads multiple accelerometers and transmits these data to one receiver element. This implementation has an upper limit on 5 sensors inside one sensor element.

Sensor data processing can be done in either the receiver element or in the sensor element. For various reasons it can be reasonable to implement some sensor data processing in the sensor element. This thesis looked into how much processing it is suitable to carry out inside these sensor elements. The star based motion capture system had advantages compared the peer-to-peer system when performing processing within the sensor elements.

Preface

This thesis is a part of my Master Degree at the Department of Informatics in the program for Electronics and Computer Technology at the University of Oslo. It is carried out in a research group called Robotics and Intelligent Systems (ROBIN).

This Master Thesis is a part of the Sensing Music-Related Actions project, which is a collaborative research project between ROBIN and the Department of Musicology at the University of Oslo. This research project are going to explore action-sound couplings in order to make new *Active Music* devices. *Active Music* means music controlled by the listener. An *Active Music* device requires sensor techniques, that are able to capture complex body movements, and segmentation methods, that are able to classify body movement data.

I would like to thank to my supervisors Jim Tørresen, Mats Høvin, Alexander Refsum Jensenius for the support during implementation and writing of this thesis. And finally, thanks to my family for help and support during the process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sensing Music-related Actions (SMRA) is a collaborative research project between the department of Musicology and the Department of Informatics at the University of Oslo. The project are going to develop new methods and technologies for sensing music-related actions and is underlying the term *Music Technology*. These methods will be used in new multimodal media devices.

The relationship between actions such as body movement and sound is called *Action-Sound* couplings. People move to music in an intuitive way. The research project is about sensing and analyzing these movements and use this knowledge to make better *Action-Sound* couplings in media players.

1.1 Research and questions

This section describes some terms and questions needed to understand the scope of this thesis and the research project it is a part of.

1.1.1 Terms

The following terms gives an insight in *Music Technology* that exploits motions in the creating of sound and music.

Action-sound couplings

Action Sound couplings are the relationships between actions, for instance body movement and the resultant sound [15]. Understanding these couplings is important to create better *Action-Sound* couplings in electronic devices.

Mapping action to sound is important in the development of new digital musical instruments.

Active Music

Present musical instruments and media players result in passive listeners. The music which is heard is static and can't be affected by the listener. When listening to music, the listener obtains an subjective reaction derived from the feelings which the music provides. New research and

technologies tries to implement these reactions in the process of creating music. The reactions which occur while listening to a musical piece can be used to manipulate the musical piece. The music is then adopted to fit the reactions and feelings of the listener. This technology can for instance be used in mp3 players. The tempo of a song can be adjusted to fit the tempo in which the listener is walking.

Music which is controlled by the listener is called *Active Music*[10]. Two terms can be used in an *Active Music* setting:

- The active listener is a person who influences the music he is hearing, for instance adjusting the rhythm of a musical piece according to the rhythm which the listener walks.
- The performer is a person who uses *Active Music* technology to create a musical piece based on his motions, this can for instance be a drum synthesis based on the measured motions.

Sensor technology is often used in modern *Music Technology*. *Active Music* devices use sensor technology to capture the semantics of a listener's movements. The resulting sensor data is used to create music.

Active Music devices

New music technology and research aim to develop more intuitive digital music instruments. Acoustic music instruments, like a guitar, give the performer feedback trough vibration. Present digital music instruments are passive and unintuitive, where sound creating parameters can't be controlled in the same way as in acoustic instruments.

A motion capture system can be used to control parameters in a sound synthesis. This requires an intuitive mapping between the motions and the resulting sound. Sensor data can be a binary string which contains the nature of the measured movements. A challenge is to extract hidden meanings from these streams of motion data. Simple examples can be extracting acceleration of the movements and use this to control a sound synthesis. Then the sound which is heard will be a function of the acceleration of the movements.

1.1.2 Methods and questions

The main research question of the SMRA project is how to exploit Action-Sound couplings in human-computer interaction [15]. The following questions need to be answered:

- Which sensor technologies can capture body motions in the best way?
- Which machine learning techniques can be used to interpret important data from a stream of body movement data?
- How is action and sound coupled in music and everyday life?
- How to control music based on the listener's movements?
- How to control sound and music using these technologies?

Figure 1.1 show how listening leads to movement and how movement leads to music. This thesis is underlying the mapping part, where movement registrations have to be able to control further musical synthesis.

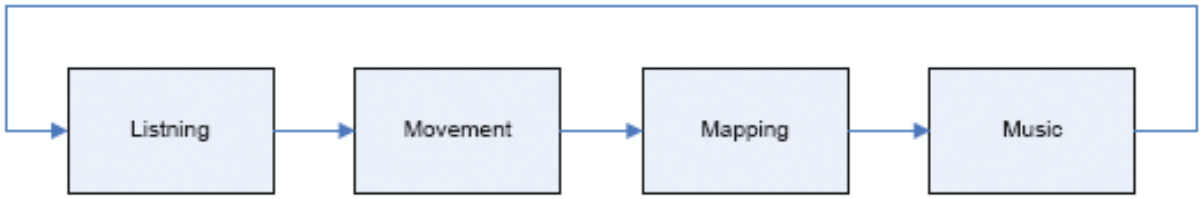


Figure 1.1: How listening and motions are connected

1.1.3 Movement registration

Motion and listening influences each other and the main goal is to figure out how. To understand how the listener's motions relate to music, motion information has to be captured, stored and analyzed. Sensor technologies like accelerometers, gyroscopes and bio sensors can be used to capture complex movements. The captured movement data can contain important information, which has to be interpreted in order to gain new knowledge on how the movements are affected by music.

There are lots of considerations concerning a motion capture system. Synchronization and timing are essential in order to get trustable data. The motions which are to be captured shall not be influenced by the sensor system. A large and heavy sensor system will influence the movements and corrupt the captured data. During capturing and transmission of movement data, low power technology is essential. For instance in a wireless sensor system, the battery size and weight points to the need for low power hardware.

1.2 Thesis overview

A motion capture system can be used to read sensor data and transmit these to applications for storing, monitoring and/or processing. The sensor data can be used for parameter control in various applications, research or data logging. Each of these applications has demands such as security, low latency, power consumption and synchronization. This thesis will look at methods for collecting sensor data for *Active Music*, using low rate[1], low power radio transceivers and low power microcontrollers. During this master thesis, different network topologies, which describe different interconnections between wireless nodes, will be implemented and tested in order to find the best way to gather sensor data.

1.2.1 Implementations

The implementations in this thesis will consist of sensor elements and receiver elements which are described below. A high level presentation of a motion capture system is shown in Figure 1.2.

[1] low transfer speeds

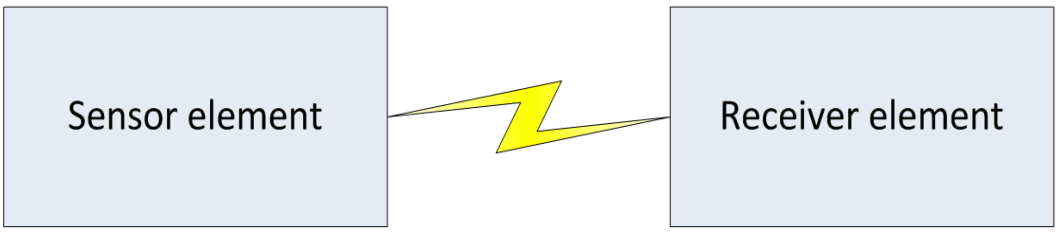


Figure 1.2: A motion capture system

The sensor element

A sensor element (Figure 1.2) is a device in a motion capture system which consists of one or more sensors and a solution for wireless transmission. This element has to be able to respond fast to changes in the incoming sensor data and transmit the sensor data with low latency.

The receiver element

The receiver element (Figure 1.2) receives sensor data from the sensor element and handles storing, monitoring and additional processing. The receiver element consists of hardware, such as microcontrollers and radio transceivers, as well as computer applications and solutions for communication between hardware and software.

Wireless transmission

The implementations in this thesis will be based on the IEEE 802.15.4 standard and the Zigbee protocol. The IEEE 802.15.4 standard is a low rate wireless personal area network (LR-WPAN) which the Zigbee protocol is build upon. Zigbee is designed for low rate application where long battery life is essential.

In order to gain knowledge about the suitability of the IEEE 802.15.4/Zigbee standard in emphActive Music applications, a wireless network setup will be implemented and different network topologies will be tested. The generated streams of body motion data will be sent through the most suitable wireless network solution and into a computer application.

A computer application, such as a terminal application or a programming environment such as MAX/MSP, will be used to monitor the captured motion data.

System requirements

A wireless motion capture system that uses body mounted sensor elements has weight and size demands. A heavy sensor element will interfere with the motions and corrupt the measured motion data. Small and light batteries are required, which raises the need for low power radio transceivers, low power microcontrollers and low power sensors.

Timing and synchronization is essential in order to use sensor data in real time applications. Low latency is important in real-time applications where movements are to be registered.

In a motion capture system there are many sources of latency. A microcontroller's analog-to-digital converter (ADC) converts one analog value at a time. The time consuming ADC conversions result in latencies between the different channels. The radio transmission and the

final communication between the hardware devices and a computer application also lead to latencies.

1.3 Problems

Low rate, low power wireless standards have limitations when transmitting high amounts of data. They also has limitation when it comes to distance, number of nodes and number of channels. This thesis will look into these limitations by answering the following questions:

- How much will these limitations influence a wireless motion capture network for real-time *Active Music* applications?
- How large sensor system is possible to implement, given the timing requirements of different *Active Music* applications?

A microcontroller will be used as the brain in the sensor elements. The microcontroller can read and process sensor data, and finally communicate with a radio transceiver for wireless transmission. A microcontroller has different features for reading external signals and can be programmed to do additional processing of sensor data. This processing can be calculating the average of a number of sensor measurements, or use a threshold that excludes unnecessary data which for instance occurs when the sensor isn't moving.

- Is the IEEE 802.15.4/Zigbee protocol suitable for a motion capture system?
  - The IEEE 802.15.4/Zigbee standard has a raw transfer rate around 250Kbps. To be able to fully take advantage of this data rate, the user data payload[2] has to contain as much data as possible. The protocol headers and tails are included in the given transfer rate. The transfer rate will be decreased, if the user data payload only contains a small number of bytes.
- What is the relation in number of accelerometers or other sensors read by one microcontroller and the resulting transmission latency?
- How much processing is preferred to do in the sensor element in order to increase the overall efficiency of the motion capture system?
- Where is it desirable to process the sensor data?
  - This question deals with the fact that processing can be done in the sensor element, in a computer application, or in a combination of those. The term processing can for instance be reducing the amount of raw sensor data. Is it desirable to do this processing in the receiver element? This involves transmitting all values to a computer as raw sensor data. This will increase the amount of data transmitted across the wireless link. Is it desirable to process sensor data in the sensor element? Sensor element processing means integrating sensor data calculation and filtrating, inside a microcontroller. This will reduce the amount of data to be transmitted across the wireless link.

[2]The amount of bytes in an IEEE 802.15.4 frame that can contain user data

- Will the number of accelerometers, read by one microcontroller, influence the need for either sensor element processing or receiver element processing?

  - A microcontroller which reads 4 accelerometers uses 12 ADC channels. The ADC measures one channel at a time, which means there will be a latency between the different channels.

    Are there a number of accelerometers which desires processing in the receiver element instead of processing in the sensor element? Are there a number of accelerometers which desires processing in the receiver element instead of processing in the sensor element?

- What kind of sensor element processing is suitable?

  - The microcontroller can measure thousands of external values each second. In this case these values are information about motions. These values can be filtred in order to decrease the amount of data. The sensor data will also contain information which hides motion patterns that can be used to understand the nature of the movements. There are some possible actions that can be performed by the microcontroller:
    * Statistic calculations such as average
    * Calculations which provides additional information based on the measured sensor data
    * Remove unnecessary data
    * Simple algorithms for pattern recognition
    * Beat detection

- What kind of computer application processing can be used?

  - Software applications such as MAX/MSP offer lots of tools which can be used to process the raw sensor data.
    * Statistic calculations such as average
    * Calculations which provides additional information based on the measured sensor data
    * Complex algorithms for pattern recognition
    * Beat detection

1.4 Practical work

This master thesis involves the following practical work.

- Hardware programming using the C-language and sensor element design

  - Programming a microcontroller for multiple sensor data measurement using the ADC converter.
  - Setting up hardware and software for IEEE 802.15.4/Zigbee transmission

  - Setting up a connection between a microcontroller and a software platform, such as MAX/MSP, using UART communication
  - Sensor data processing
  - Estimating power consumption
  - Measuring latency
- Computer application
  - Monitoring sensor data with a computer application such as MAX/MSP
  - Sensor data processing
  - Measuring latency

Implementing a wireless application such as this sensor system is a challenge. A microcontroller which reads sensor data has to be able to communicate with a radio, and this communication has strict timing requirements, which are necessary to obtain a secure communication between the radio nodes.

1.5 Outline

- Chapter 1 contains an introduction to this master thesis and an introduction to the research project it is a part of. In addition, terms needed to understand the scope of this thesis is explained.
- Chapter 2 contains a description of the hardware and software needed to implement a wireless motion capture network. This includes wireless network protocols, microcontrollers, radio transceivers and computer applications.
- Chapter 3 covers the implementation and research done in this thesis.
- Chapter 4 describes the results from the experiments done in Chapter 3
- Chapter 5 contains a conclusion.
- Chapter 6 contains suggestions for improvements and further developments.

Chapter 2

Background

This chapter will describe different hardware systems, software applications and communication protocols needed to implement a wireless motion capture system. This thesis focus on the development of low power solutions for motion capture and the implementation can be divided into a hardware part and a software part.

The hardware part of the system will handle sensor reading, processing and wireless transmission of sensor data. Atmel offers low power microcontrollers and radio transceivers which are designed for IEEE 802.15.4. They also offer firmware and example code for their products and is therefore a suitable choice for the hardware part.

The software part will handle storing and monitoring of sensor data. The MAX/MSP computer application is often used in musical applications using data from various sensors, and is therefore a suitable choice for the software part.

The hardware/software communication can be solved by using different communication protocols. This thesis will simplify this communication by using serial transmission between the computer application. An USB or an Ethernet hardware/software communication will probably increase the performance of the system but is considered to be too time consuming to implement in this thesis. A serial connection is a simple alternative that is easy to implement.

2.1 MAX/MSP/JITTER

MAX/MSP/JITTER is a computer application used for music and multimedia approaches. It consists of 3 elements:

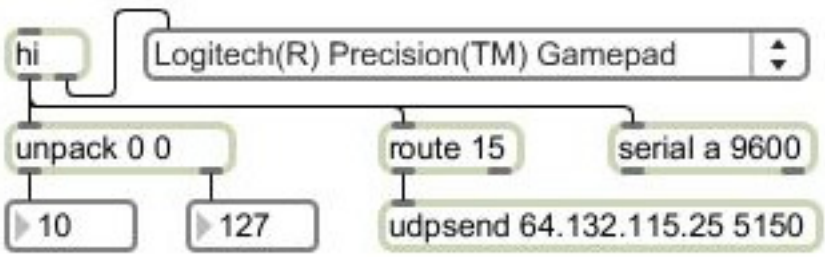


Figure 2.1: A MAX/MSP/JITTER patch

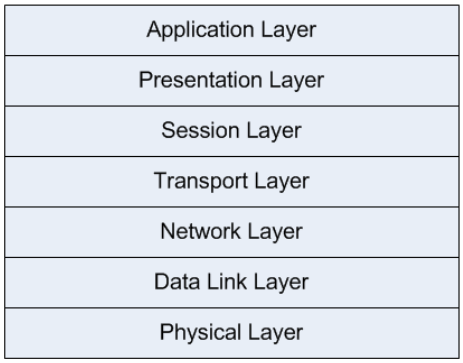


Figure 2.2: The seven layers of the OSI model

- MAX is the visual programming language which provides a graphical user interface and different kinds of communication.
- MSP provides real-time audio synthesis and digital signal processing (DSP).
- Jitter provides real-time manipulation of video and 3D graphics.

The MAX/MSP/JITTER platform is object-based where a visual interface allows the user to connect objects together and make different applications in so-called patches. Figure 2.1 shows a MAX/MSP/JITTER patch.

2.2 Open System Interconnection model

The OSI model is a reference model for data communication. It is an important model for understanding and developing of wireless networks with different characteristics. This model defines seven different layers, shown in Figure 2.2. These layers are needed to describe networks and network applications. The seven layers describe every aspect of data communication, from physical signal transmission to applications.

This is a description of the seven layers of the OSI model [20]:

Application Layer

The Application Layer provides applications in which software/user application can employ. This means that both the user and the Application Layer communicate with the same software application. The HTTP protocol works at the Application Layer. User interfaces, such as email applications and web browsers, use the HTTP to gain the information in which the user requires.

Presentation Layer

The Transport Layer is used for encryption, translation and compression, and is not always included in a network implementation. This layer is used to translate received data to fit the demands of the Application Layer, and also translate transmitted data from the Application Layer to the needed network format.

Session Layer

The Session Layer deals with connection between computers and software applications, for instance allowing a dialog between different softwares and different computers.

Transport Layer

The Transport Layer provides error detection and flow control between end points of the network connection. Some protocols are state and connection oriented, this layer makes sure the network recovers from failed transmissions and retransmit failed packets.

Network Layer

This Network Layer determines the path in which the data will be transmitted. This layer provides the IP address. A router works at this layer. It enables the data to reach its destination by jumping from router to router.

Data Link

This layer is divided into two sub layers: Logical Link Control (LLC) and Medium Access Control (MAC). The LLC is the upper sub layer in the Data Link Layer and deals with multiplexing and de-multiplexing of data transmitted across the MAC layer. When multiple terminals or network nodes is to communicate over a multipoint network, the MAC sub layer provides addressing and channel access control. Each device has a unique MAC address, which the Data Link uses to make sure that the data reach its destination. The MAC address is added to the frame which is put together by the 5 upper layers.

Physical Layer

The Physical Layer (PHY) describes the physical data transmission, such as electrical and physical specification. This can be signal specifications, voltage specifications, pin configurations and wire descriptions. In radio communications, this layer specifies the radio frequencies and radio ranges.

2.3 Wireless Personal Area Networks

Wireless Personal Area Networks (WPAN) is used to transmit information over short distances. WPAN allows small, low cost and power efficient solution for an application such as home automation, process control and energy monitoring. IrDA, Bluetooth and Zigbee are WPAN technologies.

Different topologies can be used in a WPAN:

Peer-to-peer topology

The peer-to-peer topology is a possible solution for a motion capture network. In a peer-to-peer topology, a device can communicate directly to any other devices within its range, see Figure 2.3b. This solution can be used to capture movements on specific body parts. For instance

how a wrist moves. The peer-to-peer solution can also be expanded into a multiple peer-to-peer connection, with multiple sensor elements.

Star topology

A star topology, shown in Figure 2.3a, is a possible solution for a motion capture system. A star topology enables multiple devices to talk to the same coordinator. The coordinator has the responsibility to synchronize the network and avoid data collisions.

Tree topology

In a tree network, see Figure 2.4b, a coordinator has devices connected to it. These devices can be routers which can in turn have other devices connected to them.

Mesh topology

In a mesh network, shown in Figure 2.4a, each node can transmit data and receive data from any other nodes in the system.

2.3.1 IEEE 802.15.4

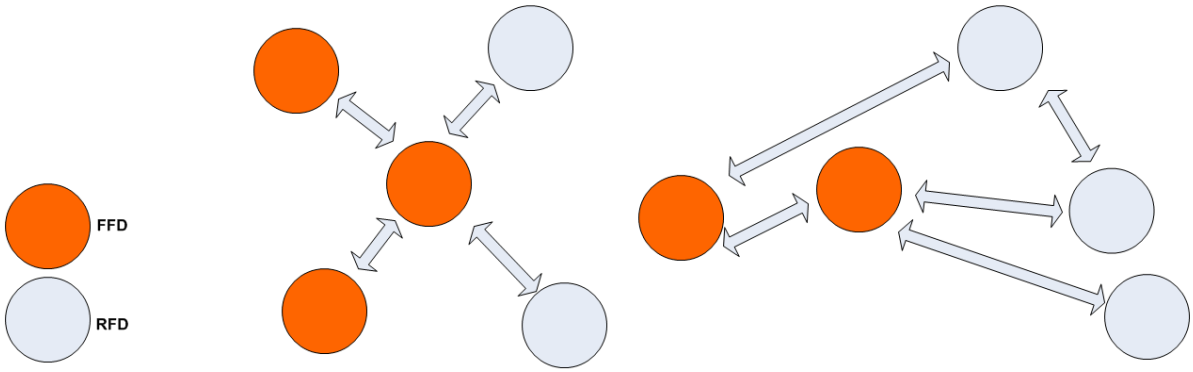
IEEE 802.15.4 is a standard designed for LR-WPAN. It provides low rate, low-cost and low power wireless networking and it can be used in motion capture systems. The IEEE 802.15.4 standard only exploits the PHY layer and the MAC layer (Data Links sub layer) which is two lowest layer of the OSI model as described in Section 2.2. These layers define the topologies and physical characteristics for low power devices operating in a typical space of 10 m. It provides 250kbps and 16 channels in the 2,4 GHz band [12].

Devices

The IEEE 802.15.4 standard provides two different devices: a Full Function Device (FFD) and a Reduced Function Device (RFD). The RFD is intended for simple application and sleeps most of the time. This can be a sensor node, which wakes up, reads sensors, transmit a frame with the sensor data and go back to sleep. A FFD can be a Personal Area Network (PAN) Coordinator, a coordinator or a device [12]. The PAN coordinator is responsible for coordinating the entire network and chooses a PAN ID which all the devices in the network exploit. It has to be awake in order to receive data whenever a device or a coordinator is transmitting. A FFD can talk to other FFD's and RFD's while the RFD can only talk to a FFD.

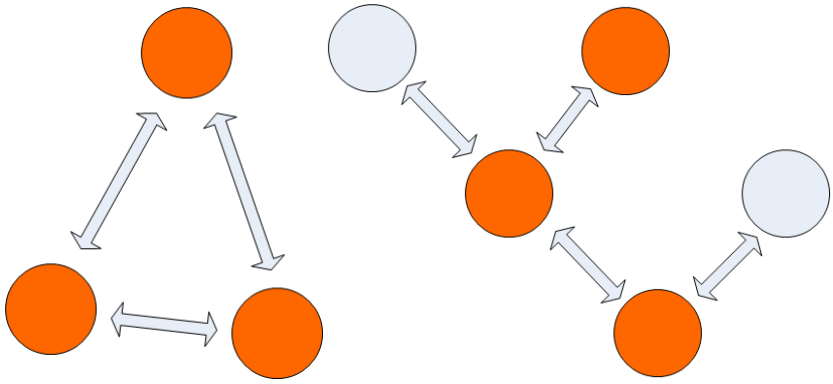
Topologies

The IEEE 802.15.4 standard provides two different topologies: a peer-to-peer topology and a star topology. More complex topologies like mesh and tree topology can be implemented by expanding the IEEE 802.15.4 standard to a full Zigbee stack.



(a) Star network (b) Peer-to-peer network

Figure 2.3: Star and peer-to-peer networks



(a) Mesh network (b) Tree network

Figure 2.4: Mesh and tree networks

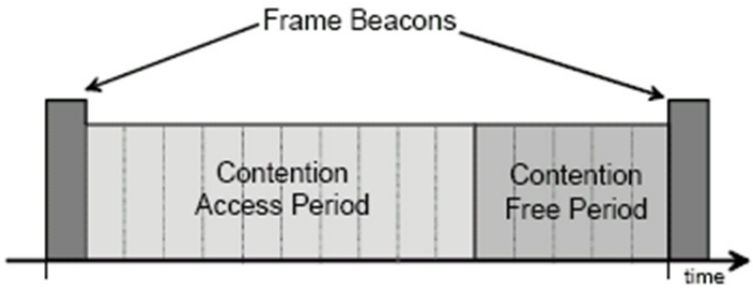
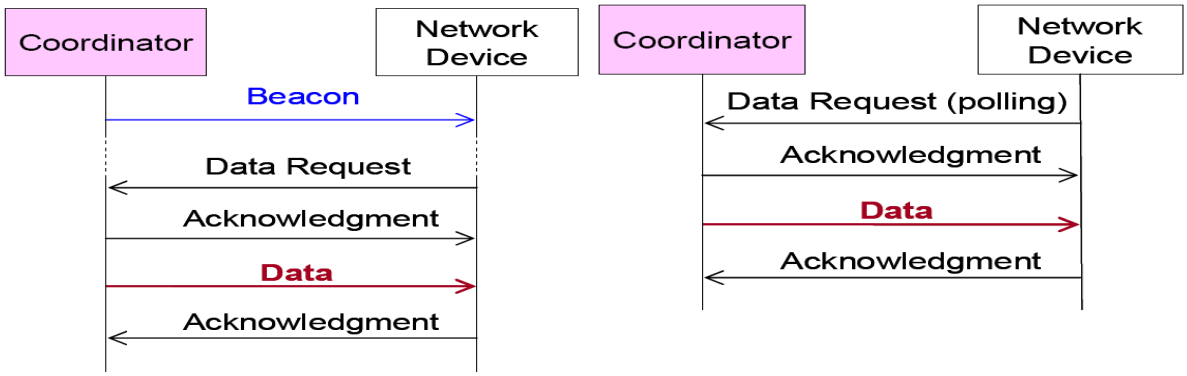


Figure 2.5: The IEEE 802.15.4 superframe



(a) Data transmission in Beacon enabled network (b) Data transmission in non-Beacon enabled network

Figure 2.6: IEEE 802.15.4 data transmission

Beacons and Superframes

Synchronization and security is essential in bigger networks where multiple nodes are transmitting. The IEEE 802.15.4 standard allows the use of Superframes. The Coordinator transmits Beacon frames, which includes control information such as synchronization, PAN identification and the Superframe structure. The Beacons help devices associating with a network. The Superframes as bounded by these Beacons. The Superframe is divided in 16 slots and consists of a Contention Access Period (CAP) and a Contention Free Period (CFP) as shown in Figure 2.5. During the CAP, transmitting devices competes using the Carries Sense Multiple Access with Collision Avoidance (CSMA/CA). The CFP offers a guaranteed time slot (GTS), which dedicates timeslots to specific applications. These applications does not compete using CSMA/CA or any other collision avoidance routines.

CSMA

In a CSMA, a device which is about to transmit, has to listen to the predefined channel for an amount of time to be sure there is no activity on the given channel. The device can transmit if the channel is sleeping, if not, it has to wait [19].

In a CSMA/CA, the device transmits after telling the other devices not to transmit. The CSMA/CA provides two different mechanisms which is slotted and unslotted. Unslotted means non-Beacon enabled, see Figure 2.6b [12]. A device waits for a random backoff period, and transmits if the channel sleeps. If the channel is busy, it waits for another random backoff period.

Slotted means Beacon enabled, see Figure 2.6a [12]. Backoff slot are included in the start of the beacon transmission. A device locates a backoff slot, than waits for a random time and transmits the data. If the channel is busy, it waits for another random time. If the channel is sleeping, it transmits at the next available slot.

IEEE 802.15.4 frames

The IEEE 802.15.4 standard offers 4 types of frames: beacon frames, data frames, acknowledgement frames and mac command frames (data request). Acknowledgements and Beacon

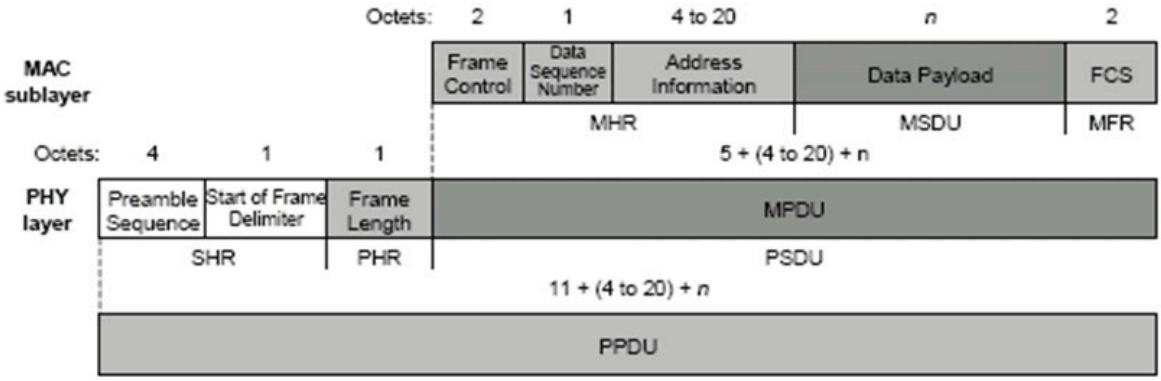


Figure 2.7: An IEEE 802.15.4 frame

frames does not use CSMA/CA. An IEEE 802.15.4 frame is shown in Figure 2.6a [10]. PPDU means PHY Protocol Data Unit and MPDU means MAC Protocol Data Unit (Section 2.2).

IEEE 802.15.4 PHY layer

The PHY is responsible for activation and deactivation of the radio device. Data transmission and reception. The physical layer handles the following tasks [4]:

- Activation and deactivation of the radio device
- Energy detection (ED) within the current channel
  - This is an estimate of signal power within the different channels
- Link quality indication (LQI) for received packets
  - This is an indication of the quality and strength of the received packet
- Clear Channel Assessment (CCA) for CSMA/CA
  1. CCA report a busy medium when noticing energy above the threshold given by the ED
  2. CCA report a busy medium when noticing a signal with the modulation characteristics of the IEEE 802.15.4 standard with energy above or below the threshold given by the ED
  3. CCA report a busy medium when noticing a signal with the modulation characteristics of the IEEE 802.15.4 standard with energy above the threshold given by the ED
- Channel frequency selection
- Data transmission and reception

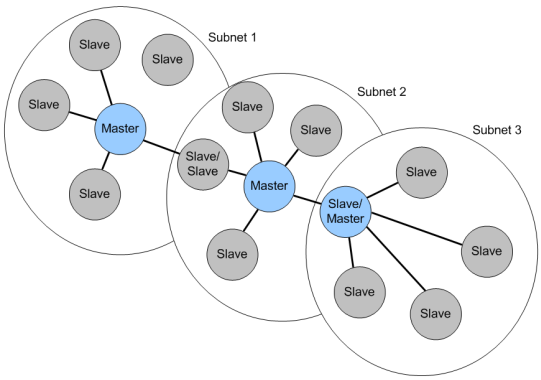


Figure 2.8: Bluetooth scatternets

IEEE 802.15.4 HAL layer

The Hardware Abstraction Layer (HAL) layer is a device specific layer which interfaces the PHY layer to a radio device. Various radios have different HAL codes, which often can be purchased from the manufacturer of the device.

IEEE 802.15.4 MAC layer

The MAC Layer provides an interface between the upper layers and the Physical Layer. This Layer enables the upper layers to access the radio. This Layer also provides a reliable link between two peers. In applications, where security is not an option, a MAC Layer can provide the necessary communication, without needing to add higher layers. The MAC layer features Beacon management, channel access, GTS management, frame validation, acknowledgements, association and disassociation [8].

Upper layers

The Application Layers and Network Layer are necessary to generate a full network standard. These layers deals with the high level software control and communicate with the MAC Layer which then perform the needed functions. Zigbee, WirelessHARD and MiWi are all based on the IEEE 802.15.4 standard and use the Application and Network layers to develop unique standards.

2.3.2 Bluetooth

Bluetooth is a WPAN technology which is often used in wireless sensor systems for musical applications. The bluetooth standard uses a star network. One master node controls multiple slaves. A Bluetooth network consists of small subnets, which can form bigger scatternets as shown in Figure 2.8. Each subnet consists of one master node and up to seven slave nodes. In bigger scatternets the subnets have one node in common. A master node in a subnet can be a slave node in other subnets.

Raw data rate	868MHz:20kb/s; 915MHz:40kb/s; 2.4GHz: 250kb/s
Range	10-20m
Latency	Down to 15ms
Channels	868MHz: 1 channel; 915Mhz: 10 channels; 2.4Ghz; 16 channels
Frequency band	868MHz 915MHz 2.4GHz
Addressing	Short 16-bit or 64-bit IEEE
Channel access	CSMA/CA and slotted CSMA/CA
Temperature	-40 to +85C

Table 2.1: Zigbee attributes

	Bluetooth	**Zigbee**
Band	2.4GHz	2.4GHz, 868MHz, 915MHz
Power	100 mW	30 mW
Target Battery Life	Days - months	6 months - 2 years
Range	10-30 m	10-75 m
Data Rate	1-3 Mbps	25-250 Kbps
Network Topologies	Ad hoc, point to point, star	Mesh, ad hoc, star
Security	128-bit encryption	128-bit encryption
Time to Wake and Transmit	3s	15ms

Table 2.2: Feature comparison of Zigbee and Bluetooth [11]

2.3.3 Zigbee

The topology is one of the differences between Zigbee and other wireless standards such as Bluetooth. In addition to the star topology, Zigbee provides mesh topology and tree topology. By halving the range of the radio, the power consumption is reduced by 75%. So by allowing mesh and tree topologies, the radio range can be reduced by adding a Router which passes along messages between an End Device and a Coordinator. A mesh network enables the dataflow to jump via several nodes before it reaches its final destination. The point to point range of Zigbee devices is limited, so by adding mesh topologies the network range increases. A Zigbee network is built up by 3 classes of nodes. At the bottom are the End Devices. These are usually RFD (Section 2.3.1), which can be a sensor or an actuator. The End devices are sleeping most of the time, and detect the nearest Router or Coordinator which it can communicates through. The routers are at the mid level and are usually a FFD (Section 2.3.1). The routers passes along messages between other devices and is always turned on. A Coordinator is also FFD (Section 2.3.1). Each network consists of one Coordinator that configures and control the entire network.

The Zigbee stack

The higher levels of the Zigbee stack handles applications such as establishing secure networks [10]. The bottom of the Zigbee protocol stack defines the physical properties.

- The PHY layer defines, as specified by the IEEE 802.15.4 standard, the radio character-

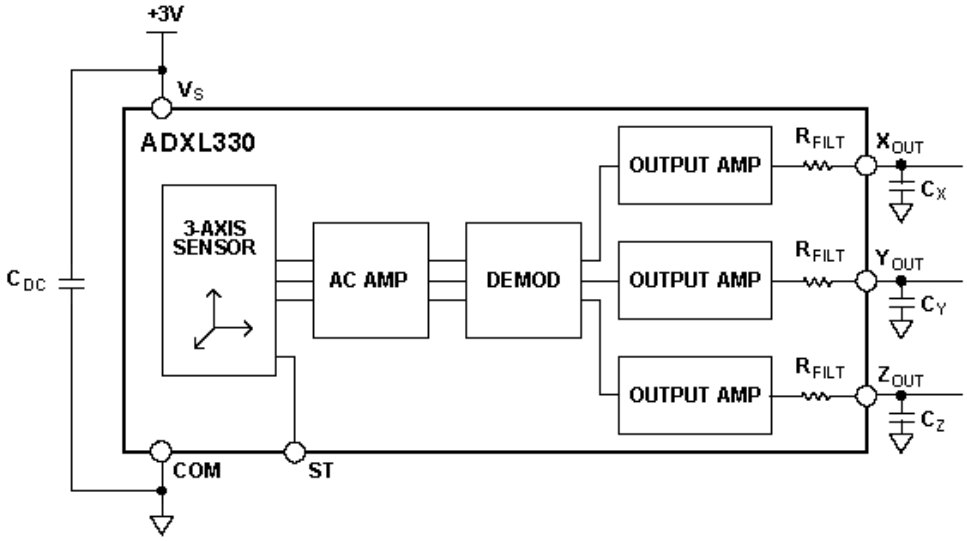


Figure 2.9: The ADXL330 accelerometer

istics such as frequency.

- The MAC layer provides, as specified by the IEEE 802.15.4 standard, a link between two devices. This means it provides a communication between two neighboring devices, and is the key element of a mesh network, where two nodes communicate via other nodes.
- The Network Layer enables the additional topologies as specified by the Zigbee standard. This layer provides the required routing needed to turn a peer-to-peer communication into star, mesh or tree networks. This layer handles the addressing, synchronization and configuring of new devices.
- The Application Layer defines the role of the device: Coordinator, Router or End Device. This layer also discovers and establishes secure relationships between network devices.

2.4 The Serial Line Internet Protocol

The Serial Line Internet Protocol (SLIP) enables transmission of IP packets over a serial line [21]. The SLIP protocol applies two characters, an ESC and an END character. The ESC character is applied between data in a frame while the delimiter is applied at the end of the frame.

2.5 Accelerometers

The Analog Devices ADXL330 chip is a low power 3-axis accelerometer. It measures acceleration with a range of $\pm$ 3 g and outputs 3 analog voltages, one for each axis. It can be used to measure static acceleration like tilt and dynamic acceleration like motion, vibration and shock. It runs at 180 μA at 1.8 V [1]. Figure 2.9 shows the ADXL330 IC.

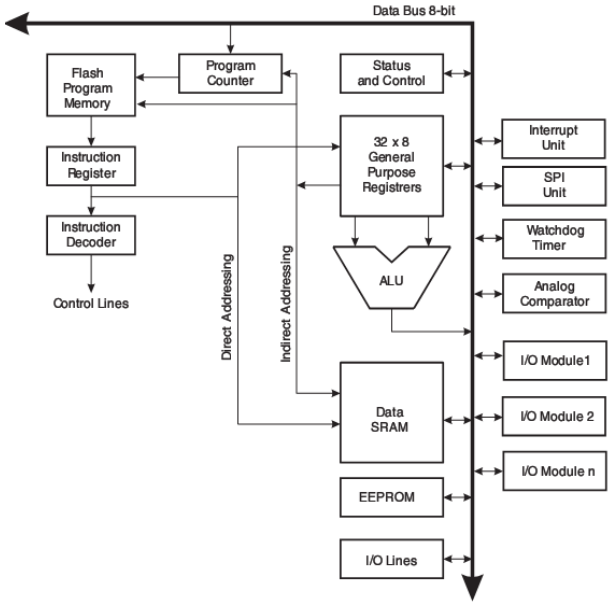


Figure 2.10: The modified RISC Harvard arcitecture

- The output voltage X_{OUT} is a function of acceleration along the x axis
- The output voltage Y_{OUT} is a function of acceleration along the y axis
- The output voltage Z_{OUT} is a function of acceleration along the z axis

2.6 Microcontrollers and the Atmel AVR

A microcontroller is a microcomputer that contains memory, I/O devices and a CPU inside an integrated circuit. FLASH, EEPROM, EPROM, ROM are integrated, which removes the need for external memory. A microcontroller has often got features like ADC, Timers, interrupts and Universal Asynchronous Receiver/transmitter (UART). The Atmel AVR microcontroller uses a modified Harvard 8-bit RISC architecture as shown in Figure 2.10. This means the program memory and the data memory are on separate busses [18]. The AVR microcontrollers were one of the first who used FLASH memory for on-chip data storage. All AVR microcontrollers have a 16 bit flash memory that can store between 1 kB to 256 kB. The flash memory can be programmed with an in-system programmer (ISP), a JTAG programmer or a high voltage parallel/serial programmer. Atmel also offers development platforms such as AVR-studio for writing, programming and debugging AVR applications. This enables the user to write instructions and application in languages like C or assembler.

2.6.1 Interrupts

A microcontroller has to be able to respond to input changes. An interrupt is an asynchronous signal which pause the main program flow, while an Interrupt Service Routine (ISR) is called. After executed the ISR, the program will continue from where it was interrupted.

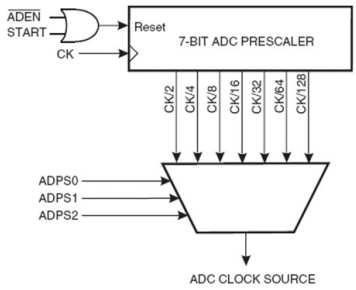


Figure 2.11: ADC prescaler

An ADC complete interrupt is called every time an ADC conversion is done. This is useful when processing has to be done after an ADC measurement. An external interrupt is for instance called when a signal is applied at one of its input pins.

2.6.2 Analog-to-digital converter

A microcontroller often contains an ADC. This is a feature that enables voltage measurement. Microcontrollers can have different input voltage ranges and digital output ranges. The digital output range is often expressed as bits. Most microcontrollers have either 8-bits or 10-bits. These bit values gives the number of values it can measure over a range of analog input values. An 8-bit can provide outputs from 0 to 255, and a 10-bit can output values from 0 to 1023. The ADC resolution can be adjusted to fit the purpose. In a wireless motion capture system, higher resolutions increase the amount of data to be transmitted. Higher resolutions also increase the conversion time. In *Active Music* applications where sensor data will be used in complex algorithms, low ADC resolution has to be avoided. Table 2.3 shows ADC values at two different resolutions. Equation 2.1 shows how to calculate the analog voltage based on the measured ADC value [6]:

$$V_{in} = \frac{V_{ref} * ADC}{\text{ADC Bit Resolution}} \tag{2.1}$$

A microcontroller can only measure one voltage at a time, but microcontrollers often got multiple ADC channels. The value in the ADMUX register determines the ADC channel. This value can be increased after each conversion which enables multi channel ADC measurements.

The ADC prescaler, shown in Figure 2.11, generates the preferred ADC clock frequency. The division factor between the microcontrollers system clock f_{osc} and the ADC clock is set by the ADPS bits in the ADCSRA register. Equation 2.2 calculates the ADC clock.

$$ADC_{clock} = \frac{f_{osc}}{ADC_{prescaler}} \tag{2.2}$$

2.6.3 Universal Asynchronous Receiver/transmitter

The UART is used to enable serial communication between a microcontroller and other devices. Serial communication means sending multiple data bits over a serial line. Synchronously UART is called USART. The baud-rate is the UART/USART transfer rate in bit per second.

10-bit			8-bit		
Analog	**Binary**	**Digital**	**Analog**	**binary**	**Digital**
5V	1111111111	1023	5V	11111111	255
4V	1100110011	819	4V	11001100	204
3V	1001100110	614	3V	10011001	153
2V	0110011010	410	2V	01100110	102
1V	0011001101	205	1V	00110011	51
0V	0000000000	0	0V	00000000	0

Table 2.3: ADC resolution

The baud-rate is given by the value in the baud-rate register UBRRn. Equation 2.3 gives the baud-rate value based on the value in the UBRRn register.

$$BAUD = \frac{f_{osc}}{16 * (UBRRn + 1)} \tag{2.3}$$

2.6.4 Timers

A microcontroller often offers a set of timers. The AVR family uses 8-bit and 16-bit timers. These timers can be used to enable small program delays, enable counters and generate Pulse Width Modulated (PWM) signals. A basic Timer counts up to the highest value given by the resolution, and then sets an overflow flag and resets the counter. The Timer offers a Clear Timer on Compare Match (CTC) mode. The counter value is compared to a value in the OCRnA register. The Timer is reset to zero when the counter value matches the value in the OCRnA register. This can be used to enable a Timer Interrupt which occurs every time the Timer is reset. Equation 2.4 gives the interval between each Timer Interrupt, based on the OCRnA value, the system clock f_{osc} and the division factor N:

$$Timer_{\text{Interrupt\_Interval}} = \frac{OCRnA}{\frac{f_{osc}}{N}} \tag{2.4}$$

2.7 MEGA1281v

The AVR ATmega1281v is a CMOS 8-bit microcontroller. This controller has a maximum clock speed of 16MHz, and most of its instructions completes in one single cycle [6]. It got 128 Kbyte self programming Flash Program Memory, 8 Kbyte SRAM and 4 Kbyte EEPROM. It contains typical microcontroller features like 10 bit ADC, timers, USART, SPI, PWM and analog comparator. Figure 2.12 show the pin configuration of the 100-pin ATmega1281v version.

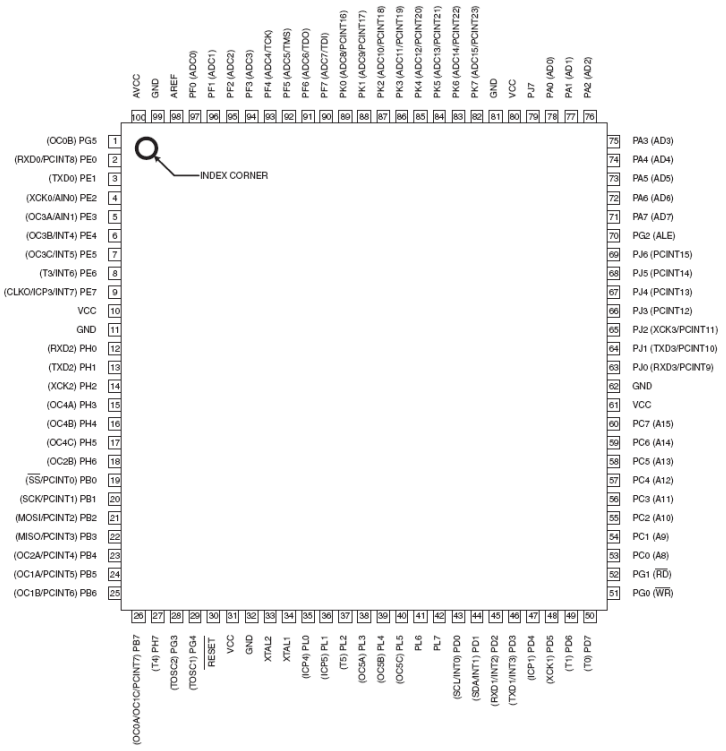


Figure 2.12: Pin configuration on the 100 pin version of ATMEGA1281v

2.8 STK500 development kit

Atmel offers various development systems and starter kits for their AVR devices. The STK500 development kit is designed to provide a simple platform for designers to do various tests on the microcontrollers. The STK500 provides leds, switches and breakout pins that allow the user to use features like ADC and SPI [3]. This board has sockets for various microcontrollers and is compatible with AVR studio. The microcontroller can be programmed with different computer applications and a JTAG interface, an ISP interface or a RS-232 serial connection.

2.9 STK501 extension board

The STK501 extension board is an extension board for the STK500. This enables the use of 64-pin microcontrollers such as ATMEGA1281 and ATMEGA103 [2]. It features a Zero Insertion Force (ZIF) socket which simplifies the use of microcontrollers in TQFP packages.

2.10 AT86RF230 and the ATAVRRZ502

AT86RF230 is a low power 2.4GHz radio transceiver, designed for Zigbee/IEEE802.15.4 applications [7]. The radio transceiver offers automatic CSMA/CA, Frame Retransmissions and Frame Acknowledgments. It runs on 1.8V to 3.6V and uses 15.5 mA when receiving, 16.5 mA when transmitting and 20 nA when sleeping. It offers a 128 byte internal SRAM needed to store received data or data to be transmitted.

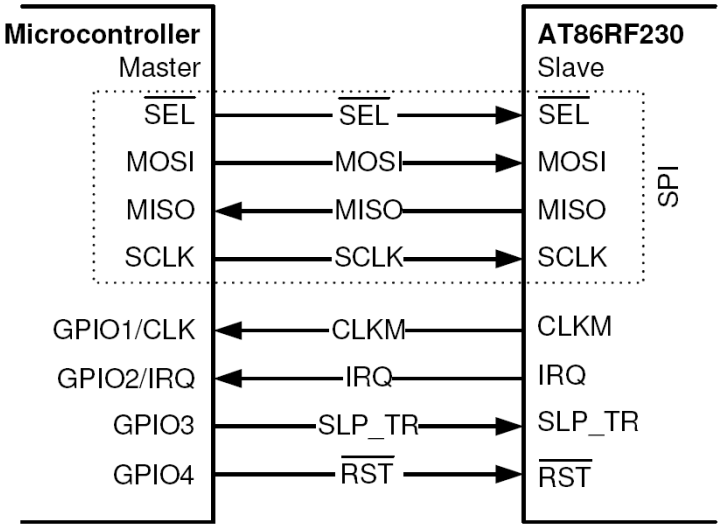


Figure 2.13: Microcontroller to AT86RF230 Interface

The RZ502 RF accessory kit is a break-out board for the AT86RF230 radio transceiver which enables the user to access the pins.

This is a SPI-to-antenna solution were all RF components are placed on chip. It requires an external antenna and a SPI connection to a microcontroller. The microcontroller is configured as a master and the radio transceiver as a slave. The microcontroller can then control the radio receiver using:

- Register read write
- Frame read and write
- SRAM read and write

The interface between a microcontroller and the radio transceiver is shown in Figure 2.13. This interface consists of a SPI interface (MOSI, MISO, SCLK, /SEL), digital control pins (/RST, SLP_TR), the interrupt pin (IRQ) and the clock output pin (CLKM) [5]. The SPI is used by the microcontroller to upload or download frames from the radio transceiver and for register access. The following control signals are connected to the microcontrollers I/O:

- The SLP_TR is a multipurpose control signal that can be used to change radio transceiver states.
  - The AT86RF230 is a state oriented device. The function is state dependent and it is controlled by an output pin of the microcontroller. A high signal on the TRX_PIN_SLP_TR causes the radio to change state. This change depends on in which state the radio transceiver was in. For instance, a frame is sent if the radio is in state PLL_ON.
- IRQ is the radio transceivers interrupt request signal
  - This is an external interrupt pin at the microcontroller, which stays HIGH, until the interrupt occurs.

Addr.	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x01	TRX_STATUS	CCA_DONE	CCA_STATUS	Reserved	TRX_STATUS					pg. 27, 38, 55
0x02	TRX_STATE	TRAC_STATUS			TRX_CMD					pg. 28, 39
0x03	TRX_CTRL_0	PAD_IO		PAD_IO_CLKM		CLKM_SHA_SEL	CLKM_CTRL			pg. 7, 69
0x05	PHY_TX_PWR	TX_AUTO_CRC_ON	Reserved			TX_PWR				pg. 49, 59
0x06	PHY_RSSI	RX_CRC_VALID	Reserved		RSSI					pg. 50, 52
0x07	PHY_ED_LEVEL	ED_LEVEL								pg. 51
0x08	PHY_CC_CCA	CCA_REQUEST	CCA_MODE		CHANNEL					pg. 56, 72
0x09	CCA_THRES	Reserved				CCA_ED_THRES				pg. 57
0x0E	IRQ_MASK	IRQ_MASK								pg. 20
0x0F	IRQ_STATUS	IRQ_7	IRQ_6	Reserved		IRQ_3	IRQ_2	IRQ_1	IRQ_0	pg. 20
0x10	VREG_CTRL	AVREG_EXT	AVDD_OK	Reserved		DVREG_EXT	DVDD_OK	Reserved		pg. 63
0x11	BATMON	Reserved		BATMON_OK	BATMON_HR	BATMON_VTH				pg. 65
0x12	XOSC_CTRL	XTAL_MODE				XTAL_TRIM				pg. 70
0x1A	PLL_CF	PLL_CF_START	Reserved							pg. 73
0x1B	PLL_DCU	PLL_DCU_START	Reserved							pg. 73
0x1C	PART_NUM	PART_NUM								pg. 16
0x1D	VERSION_NUM	VERSION_NUM								pg. 16
0x1E	MAN_ID_0	MAN_ID_0								pg. 17
0x1F	MAN_ID_1	MAN_ID_1								pg. 17
0x20	SHORT_ADDR_0	SHORT_ADDR_0								pg. 42
0x21	SHORT_ADDR_1	SHORT_ADDR_1								pg. 42
0x22	PAN_ID_0	PAN_ID_0								pg. 42
0x23	PAN_ID_1	PAN_ID_1								pg. 42
0x24	IEEE_ADDR_0	IEEE_ADDR_0								pg. 43
0x25	IEEE_ADDR_1	IEEE_ADDR_1								pg. 43
0x26	IEEE_ADDR_2	IEEE_ADDR_2								pg. 43
0x27	IEEE_ADDR_3	IEEE_ADDR_3								pg. 43
0x28	IEEE_ADDR_4	IEEE_ADDR_4								pg. 43
0x29	IEEE_ADDR_5	IEEE_ADDR_5								pg. 43
0x2A	IEEE_ADDR_6	IEEE_ADDR_6								pg. 44
0x2B	IEEE_ADDR_7	IEEE_ADDR_7								pg. 44
0x2C	XAH_CTRL	MAX_FRAME_RETRIES				MAX_CSMA_RETRIES			Reserved	pg. 40
0x2D	CSMA_SEED_0	CSMA_SEED_0								pg. 41
0x2E	CSMA_SEED_1	MIN_BE		AACK_SET_PD	Reserved	I_AM_COORD	CSMA_SEED_1			pg. 41

Figure 2.14: The AT86RF230 registers

- RST is the radio transceivers reset signal
  - The RST pin, which is active low, is used to reset the radio transceiver. This is done by an output pin on the microcontroller. This pin is set to HIGH in normal mode. A low signal from the microcontroller keeps the transceiver in reset.
- CLKM is the radio transceivers output clock signal
  - The radio transceiver can supply a clock signal through the TRX_PIN_CLKM pin. The microcontroller can use this signal as a timer or as the main clock.

2.10.1 Registers

The radio transceiver registers in Figure 2.14 is described in the AT86RF230 data sheet [7]. These are 8 bit registers which the user can access with a SPI command and can be used for reading, initializing and configuration of the radio device.

It allows the user to configure parameters like radio channel and transmit power and it allows the user to read status information, such as radio transceiver status. Finally it allows the user to initialize the transactions, which can be handled in different modes such as basic and extended. The basic mode handles receiving and transmitting of frames and power up, these features are called basic radio transceiver states. The extended mode is designed to fit the functionality required by the IEEE 802.15.4 standard, and provides features like Automatic Acknowledgement and CSMA/CA. These features are called extended radio transceiver states.

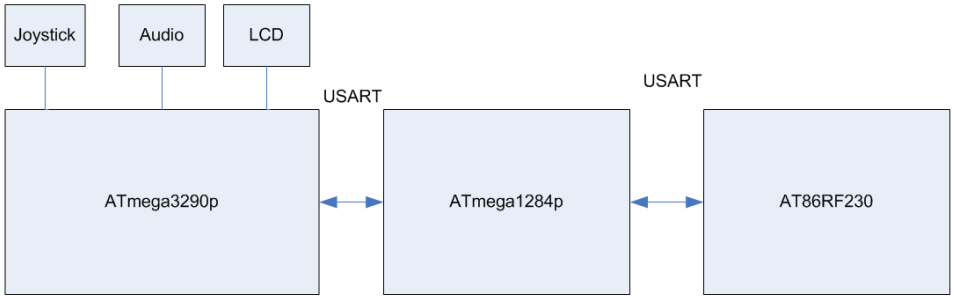


Figure 2.15: The RZRAVEN board

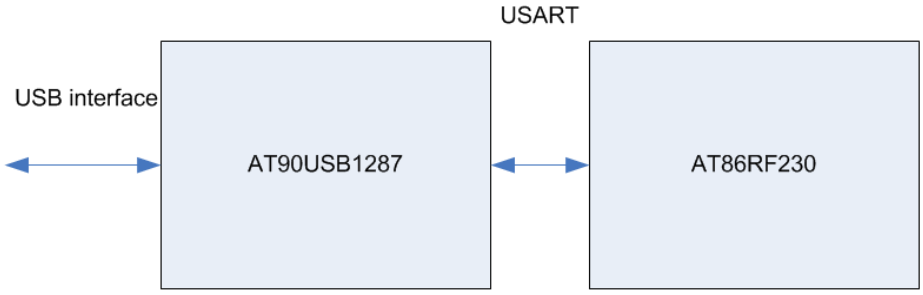


Figure 2.16: The RZUSBSTICK

An example of an extended radio transceiver state is RX_AACK_ON. An example of a basic radio transceiver state is BUSY_RX.

Sub registers are set of bits within an 8 bit register. The state of the radio receiver is determined by reading the SR_TRX_STATUS sub register.

2.11 RZRAVEN kit

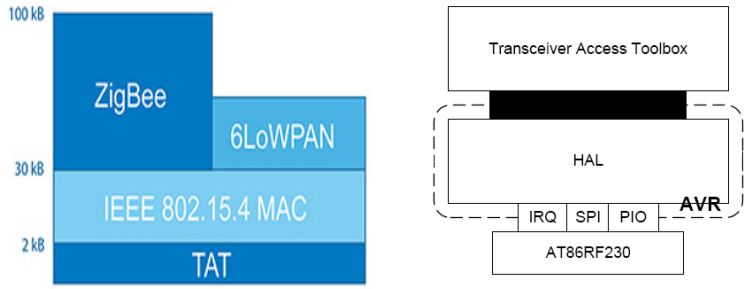
The RZRAVEN is a development kit for the AT86RF230 radio transceiver and it is intended for wireless sensor applications. It consists of RZUSBSTICK and two AVRAVEN boards. The AVRAVEN boards are wireless nodes which can communicate with each other or the RZUSBSTICK. The RZUSBSTICK is connected to a computer through an USB port.

The AVRAVEN board, shown in Figure 2.15, consists of one ATmega 1284p microcontroller which communicates with the AT86RF230 radio transceiver, and one ATmega 3290p which drives an LCD. The AVRAVEN offers a loudspeaker, microphone and a joystick which is used to control a menu. The AVRAVEN also offers various IO headers that enables the user to connect to the microcontroller's pins. This can be used for external voltage measurements.

The RZUSBSTICK, shown in Figure 2.16, consist of one AT90USB1287 microcontroller which communicates with an AT86RF230 radio transceiver and also communicates with a computer application. The communication between the RZUSBSTICK and a computer application can be implemented using USB protocol or regular USART.

A RZRAVEN kit is used in the star based motion capture system implemented in this thesis.

The AVRaven boards acts as sensor elements, while the RZUSBSTICK, together with a computer, acts as a receiver element. An accelerometer is connected to each AVRaven board's



(a) Atmel software (b) Transceiver Access Toolbox

Figure 2.17: Atmel software solution and the Transceiver Access Toolbox layers

ADC pins.

2.12 Atmel's wireless microcontroller software

Atmel offers software for their wireless solutions. IEEE 802.15.4 MAC, 6LoWPAN, Zigbee PRO stack and Transceiver Access Toolbox software are available. See Figure 2.17a. Application notes such as AT86RF230 Software Programmers Guide explains how to configure and use the features of the radio transceiver. These application notes contain examples on how to establish a communication between different radio transceivers using the Atmel wireless software.

2.12.1 The Transceiver Access Toolbox

The Transceiver Access Toolbox (TAT) is a series of low level drivers that provides access to the AT86RF230 radio transceiver. TAT is an open source code, written in C, and is implemented as a library that can be used in AVR studio. The TAT contains two layers of code. The HAL is found on the bottom, below the TAT. The HAL is a software layer that directly communicates with hardware. The TAT uses the HAL as an interface between the microcontroller and the AT86RF230 radio transceiver. HAL controls and configures the AT86RF230 radio transceiver though the SPI interface and the microcontrollers additional I/O pins.

The TAT code is radio transceiver dependent.

2.12.2 The Atmel IEEE 802.15.4 MAC layer code

Atmel has designed a MAC Layer which fits the demand of the IEEE 802.15.4 standard [4]. The MAC layer provides an interface between the physical radio channel and the upper layers. Figure 2.18 shows a modified OSI model which shows the IEEE 802.15.4 standard.

2.13 Processing methods

There are two reasons for implementing filters and other processing methods inside the microcontroller:

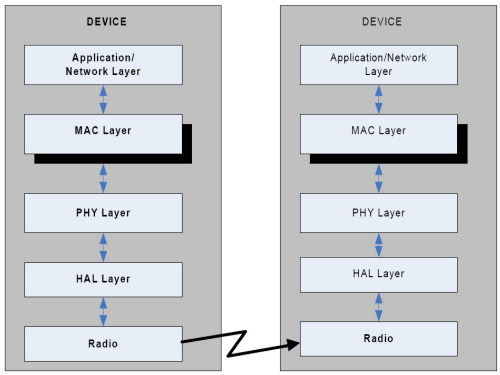


Figure 2.18: IEEE 802.15.4 standard

- Minimize the amount of data to be transferred across the wireless link
- The transmission latencies can be taken advantages of by executing additional processing inside the sensor elements

A triaxial accelerometer outputs 3 voltages. These voltages are functions of x, y and z axis. It can measure both dynamic and static acceleration. Static means measuring a constant force of gravity and dynamic means measuring a movement. The following sections describe possible processing methods for accelerometer data which can be implemented in the microcontroller.

The median filter

The median is the number which separates the lower half of a string of numbers, from the upper half. The series of numbers, which is used to calculate the median, forms a window. A non-overlapping median filter replaces the whole window with new numbers after each calculation. A running median filter uses a 'first in, first out' mechanism where part of the window is replaced by new numbers.

A median calculation is shown below. A matrix consists of a series of numbers. The rows can for instance be one on the axis on an accelerometer. These numbers are sorted, and finally the median values are gained.

$$(\textbf{unsorted}) \begin{bmatrix} 5 & 10 & 2 & 7 & 7 \\ 12 & 3 & 8 & 10 & 9 \\ 9 & 3 & 13 & 10 & 2 \end{bmatrix} (\textbf{sorted}) => \begin{bmatrix} 2 & 7 & 7 & 7 & 10 \\ 3 & 8 & 9 & 10 & 12 \\ 2 & 3 & 9 & 10 & 13 \end{bmatrix} (\textbf{median}) => \begin{bmatrix} 7 \\ 9 \\ 9 \end{bmatrix}$$

The mean is the sum of all numbers in a string, divided by the total amount of numbers. As opposed to the mean filter, a median filter can be used to remove noise spikes. These noise spikes will interfere with the results in a mean filter. Therefore a median filter is preferable as a processing tool in a motion capture system.

A median filter can be implemented inside the microcontroller. The ADC values are stored in a matrix as shown below.

$$\begin{bmatrix} ADC_{11} & ADC_{12} & \cdots & ADC_{1n} \\ ADC_{21} & ADC_{22} & \cdots & ADC_{2n} \\ ADC_{31} & ADC_{32} & \cdots & ADC_{3n} \end{bmatrix} = \begin{bmatrix} Xaxis_1 & Xaxis_2 & \cdots & Xaxis_n \\ Yaxis_1 & Yaxis_2 & \cdots & Yaxis_n \\ Zaxis_1 & Zaxis_2 & \cdots & Zaxis_n \end{bmatrix}$$

The mean filter

The mean filter calculates the average of a window of numbers.

$$(\textbf{unsorted}) \begin{bmatrix} 5 & 10 & 2 & 7 & 7 \\ 12 & 3 & 8 & 10 & 9 \\ 9 & 3 & 13 & 10 & 2 \end{bmatrix} (\textbf{mean}) => \begin{bmatrix} 6.2 \\ 8.4 \\ 7.4 \end{bmatrix}$$

The high-pass filter

A high pass filter can be implemented in order to remove data caused by small movements. The high frequency components corresponds to rapid shaking [13]. The high pass filter calculates the difference between a new measurement and the previous measurement. If the difference between two measurements is small, the data can be erased. If the difference between two measurements is large, the data is kept. A simple high pass filter is shown in the following equation [14]:

$$y[n] = (0.5 * x[n]) - (0.5 * x[n-1]) \tag{2.5}$$

The derivative filter

The derivative filter calculates the change in the signal by subtracting the former value from the present. A zero output means no movement. This can be described as a high-pass filter with a high cutoff frequency.

$$y[n] = (x[n]) - (x[n-1]) \tag{2.6}$$

The low-pass filter

A low-pass filter can be implemented in order to remove data caused by fast and sudden movements. The low frequency components correspond to tilting and rolling [13]. The low-pass filter calculates the average between a new measurement and the previous measurement. If the difference between two measurements is small, the data is kept. If the difference between two measurements is large the data can be erased. A simple low pass filter is shown in the following equation [14]:

$$y[n] = (0.5 * x[n]) + (0.5 * x[n-1]) \tag{2.7}$$

Threshold

The measured sensor data can be compared to a threshold. If the measured data is below this threshold, it will be erased. This threshold can be added to the low-pass filter or the high-pass filter in order to sort out unnecessary sensor data.

Polar coordinates

Calculate the polar coordinates which define the current orientation of the sensor with respect to the zero state (X=Y=Z=0).

- Azimuth/yaw is the rotation around the z axis of the sensor.
- Elevation/pitch is the rotation around the y axis of the sensor.
- Roll is the rotation around the x axis of the sensor.

Roll and elevation can easily be derived from accelerometer data because of the 1 G component along the Z axis given by the earth. For Azimuth calculations, other sensors or solutions like gyroscope is needed.

Chapter 3

Methods and implementations

This chapter describes different wireless motion capture system implementations done in this thesis. These implementations are based on the IEEE 802.15.4 standard. Since Zigbee is built upon the IEEE 802.15.4 standard (see Section 2.3.3), the use of this standard will also give an indication on how suitable the Zigbee standard will be in *Active Music* applications.

This chapter is divided in the following sections:

- Section 3.1 describes different motion capture system implementations that have been considered during this thesis.
- Section 3.2 describes the implementation of a peer-to-peer based motion capture system.
- Section 3.3 describes the implementation of a star based motion capture system.
- Section 3.4 describes the serial communication within the implemented motion capture systems.
- Section 3.5 describes implementations and factors considering sensor data processing within the sensor element.
- Section 3.6 describes a implementation that can be a part of an simple *Active Music* application.

3.1 Possible solutions for a wireless motion capture setup

This thesis deals with finding good solutions for an IEEE 802.15.4/Zigbee based motion capture system. A motion capture system has to be able to transfer data from multiple sensors with low latency. There are many different topologies and the number of accelerometers can vary. This section describes 3 different solutions which have been considered implemented during this thesis.

3.1.1 A peer-to-peer based motion capture system

In a peer-to-peer based motion capture system (see Section 2.3), one microcontroller reads multiple sensors and transmits sensor data wirelessly to a receiver which is connected to a computer application. This is shown in Figure 3.1. When a sensor element is reading multiple

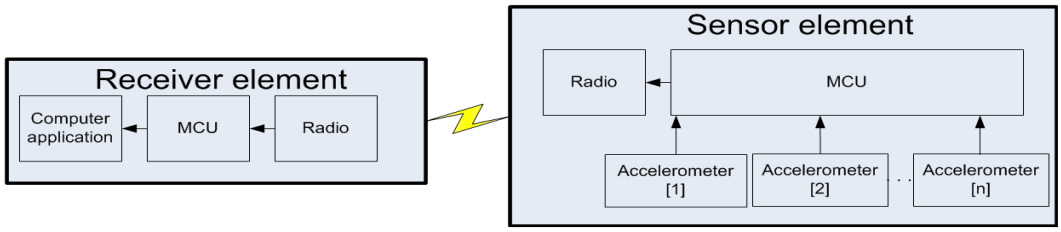


Figure 3.1: A peer-to-peer based motion capture system

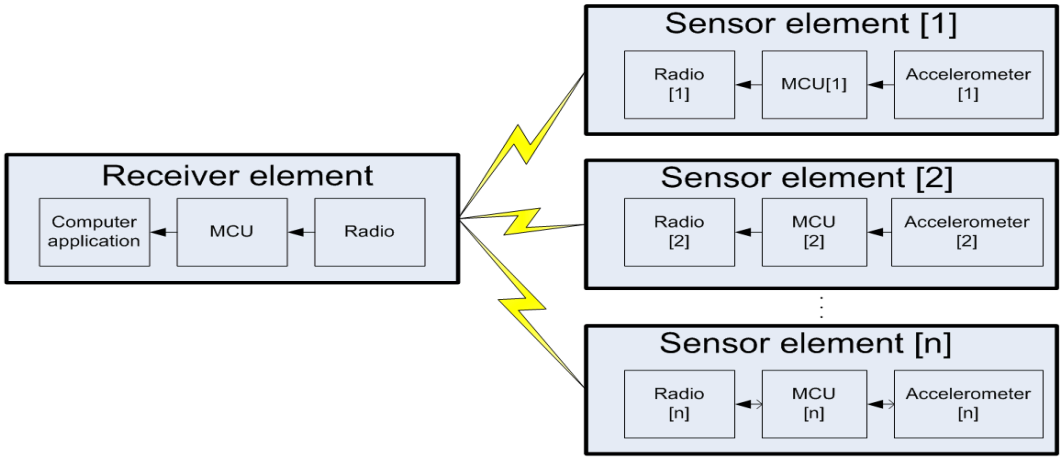


Figure 3.2: A star based motion capture system

sensors, wires are needed. Wires are unwanted because they can influence motions. The peer-to-peer connection can be expanded into a motion capture system where multiple peer-to-peer connections capture movements on different parts on the body.

3.1.2 A star based motion capture system

In a star based motion capture system, multiple sensor elements individually transfer data streams to a receiver. This is shown in Figure 3.2. Each sensor element will contain one sensor, one microcontroller and one radio transceiver. The receiver element consists of one radio transceiver and will receive data from all sensor elements. This is a star topology (see Section 2.3) where multiple elements directly communicate with a master receiver node. Figure 3.3 shows an implementation of this topology where 6 sensor elements is connected to a person, and these sensor elements communicate with a receiver element.

3.1.3 An advanced network topology implementation

Figure 3.4 shows a more advanced network topology where a radio transceiver node receives data from 2 sensor elements, and acts as a router which retransmit these data to a master receiver. The master receiver also receives data from other sensor platforms. This kind of multi hop topologies, such as mesh (Section 2.3) and tree (Section 2.3), is what makes a full Zigbee implementation (Section 2.3.3) a suitable choice.

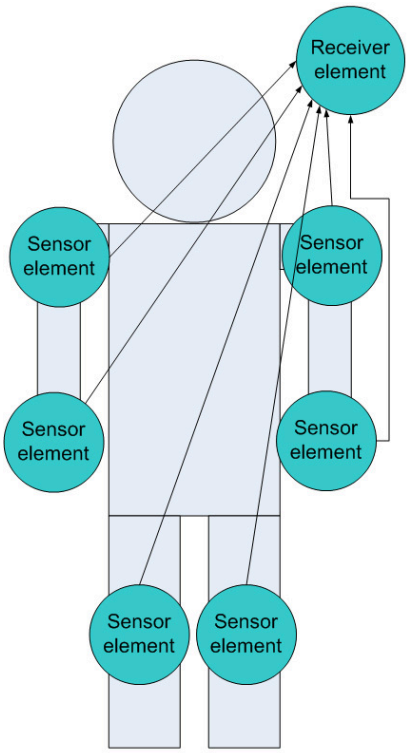


Figure 3.3: A star network example

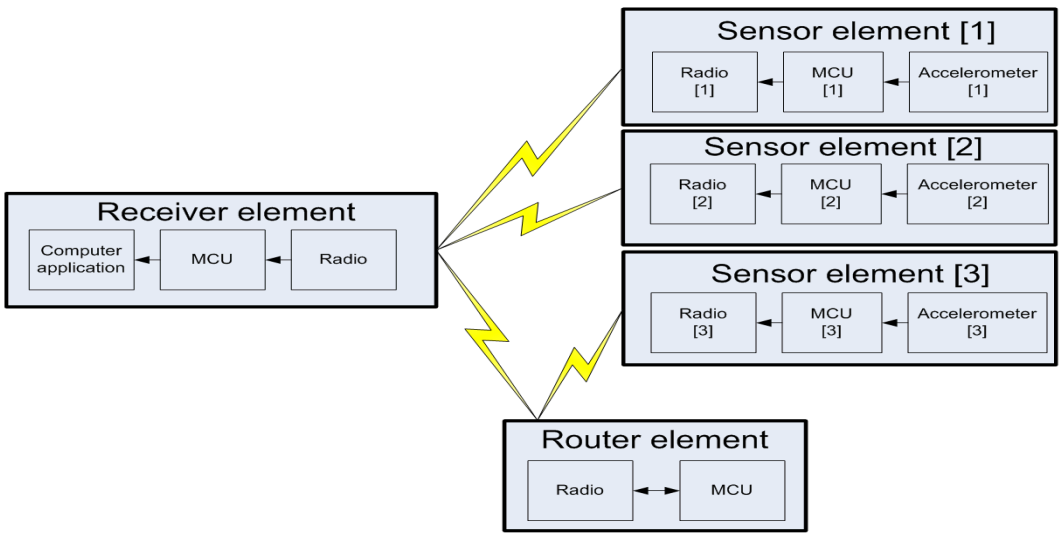


Figure 3.4: A multi hop motion capture system

3.1.4 Chosen solutions

This thesis will focus on the implementation of the peer-to-peer based motion capture system and the star motion capture system described above. Topologies like mesh and tree are hard to implement and have been considered being too complex for *Active Music* applications.

The two chosen implementations will be tested according to effective transfer rates across the wireless link and overall system latencies. The results will be compared to a latency requirement for musical applications proposed in [17]. This article proposes the acceptable upper bound for a sound synthesis' audible reaction to motions to be 10 ms. Latencies which exceeds this requirement will be noticeable for the performers.

This thesis will, based on the measured latencies in the implementations, suggest an amount of processing which can be done inside the sensor elements [1]. If the wireless transmission will result in a 5ms latency, the remaining 5 ms could be used for additional processing without exceeding the requirements in [17]. A parallel program flow [2] will not be taken into account in this thesis.

3.2 A peer-to-peer motion capture system based on the IEEE 802.15.4 standard

This section describes the implemented peer-to-peer motion capture system based on the IEEE 802.15.4 standard (see Section 2.3 and 3.1.1). This solution is most suitable in a motion capture system where a complete wireless setup isn't needed. Or in a motion capture system where multiple receiver elements individually reads multiple sensor elements.

The sensor element can be implemented with a microcontroller, a radio transceiver and one or more sensors. The microcontroller will read data from the sensors, perform the wanted data processing, build IEEE 802.15.4 frames, and finally transmit.

The receiver element consists of a microcontroller and a radio transceiver, and has the responsibility to detect and receive incomming IEEE 802.15.4 frames and transmit and monitor these data in a computer application.

This implementation will be tested according to overall system latency[3]. This will point out how suitable this setup will be in an *Active Music* setting, how many accelerometers it's reasonable to include in a sensor element and how much processing it's reasonable to perform inside the sensor element.

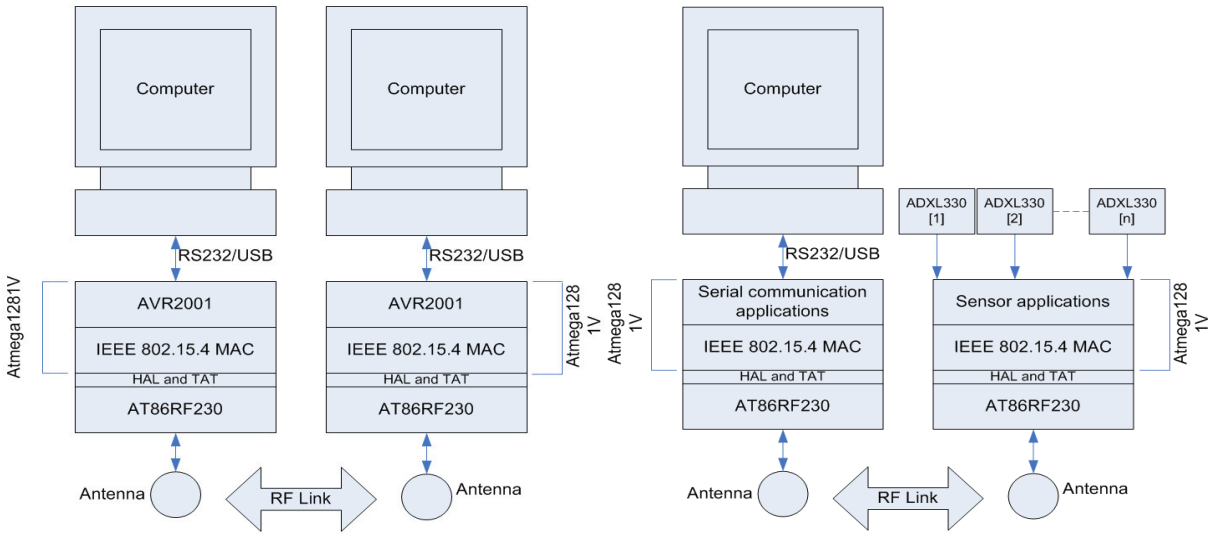
3.2.1 AVR2001

AVR2001 Software Programmers Guide is an application note offered by Atmel. This note describes the programming sequences needed to control the AT86RF230 radio transceiver (Section 2.10). It offers theoretical information and example files that users can exploit for specific Atmel AVR devices. The application note suggests the use of Mega1281v microcontroller (Section 2.7) to control the AT86RF230 radio transceiver. The STK500 board (Section 2.8),

[1] Sensor processing algorithms inside the sensor element's microcontrollers

[2] Sensor data measurements and processing could be executed at the same time as the IEEE 802.15.4 frame transmission

[3] The latency from motion to generated sound



(a) The AVR2001 peer-to-peer example (b) The modified AVR2001 peer-to-peer example

Figure 3.5: The AVR2001 peer-to-peer example

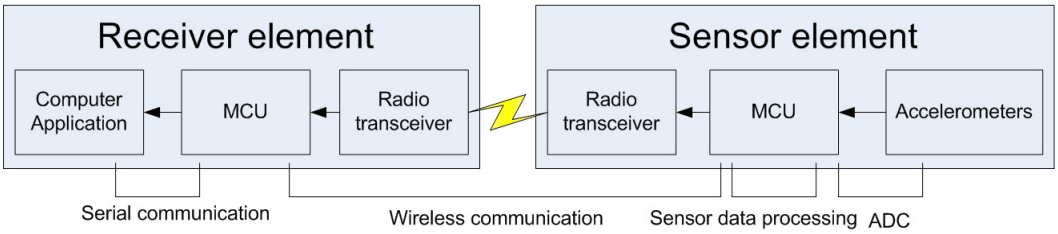


Figure 3.6: Possible latency sources in the AVR 2001 peer-to-peer example

the STK501 extension board (Section 2.9) and the RZ502 (Section 2.10), are needed to connect the radio transceiver and the microcontroller.

One of the examples in the application note is a peer-to-peer connection between two radio platforms. Two computers communicate through this peer-to-peer connection using an USART (see Section 2.6.3). Users can write sentences in a terminal application, and transmit these sentences across the wireless link. Figure 3.5a shows how the computer communicates with the microcontroller and the radio. The Atmel MAC software stack (Section 2.12.2) defines the IEEE 802.15.4 standard, while the TAT (Section 2.12.1) and the HAL layer (Section 2.3.1) provide the functions needed to use the radio transceiver.

This example provides a good starting point for a peer-to-peer motion capture system. Figure 3.5b shows a modified version implemented in this thesis. Data from accelerometers is read and sent through the wireless link and into a computer application. An application inside the microcontroller reads multiple accelerometers, process the data, builds IEEE 802.15.4 frames (Section 2.3.1) and sends those frames across the wireless link.

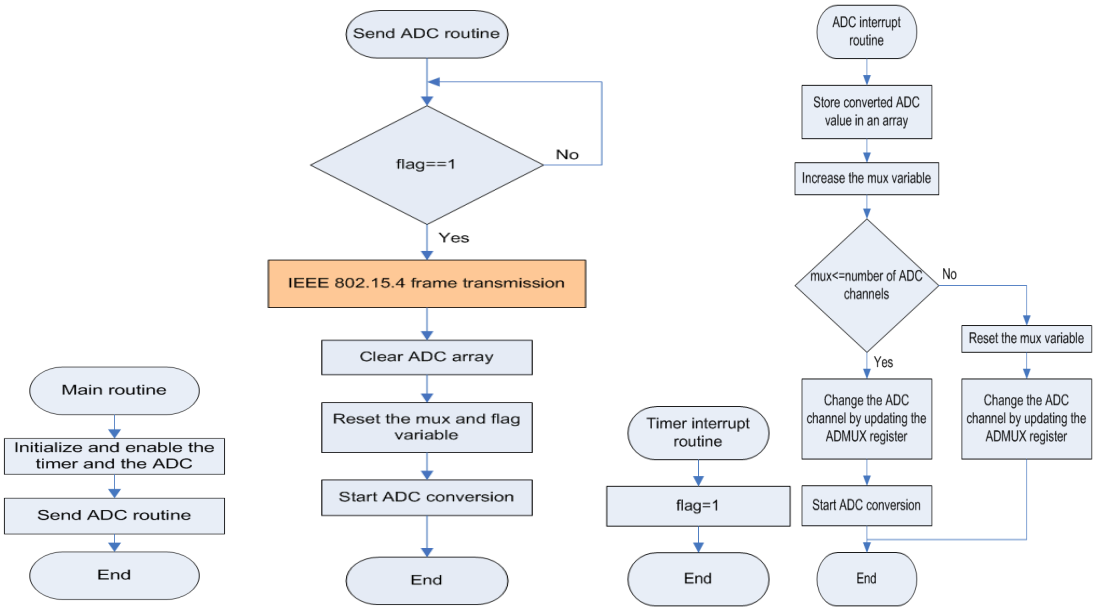
3.2.2 Latency sources

Latency is critical in *Active Music* application where motions directly affect the music. This peer-to-peer motion capture system has several latency sources. These sources are shown in

Figure 3.6. The measurements and estimates in Chapter 4 will point out which of these sources can be categorized as the bottleneck of the system and will be the main latency contributor.

- Latency sources within the sensor element
  - Sensor data processing (Figure 3.6)
    - A microcontroller can be used to filter, process and extract important information from the sensor data. In cases where the data transmission is the main latency source, a sensor element can do additional processing while it's waiting for its time slot on the wireless network. This kind of processing will increase the hardware power consumption, but will cause less processing in the computer application. In a motion capture system with many nodes, it will be preferable if each node minimize their data streams.
  - ADC conversion time (Figure 3.6)
    - The ADC conversion time is another latency source (Section 2.6.2) and the total conversion time will increase while adding accelerometers. This is mainly an issue in this peer-to-peer based motion capture system. The sensor element's microcontroller has to read multiple accelerometers. This will result in a larger ADC conversion latency compared to the star based setup where each sensor element only measure one accelerometer.
- Latency sources within the receiver element
  - Serial communication between the microcontroller and a computer application using an USART connection (Figure 3.6)
    - The USART communication has a transfer rate that will be limited by the computer application and the microcontroller.
- Latency sources between the sensor element and the receiver element
  - Wireless transmission (Figure 3.6)
    - The IEEE 802.15.4/Zigbee protocol has a 250 Kbs raw transfer rate (Section 2.3.3). This includes frame headers and tails, and will be reduced if the frames isn't filled with the maximum amount of user data. Other IEEE 802.15.4 features like Acknowledgements (Section 2.3.1) will also decrease the transfer rate.

Different latency tests are conducted on this implementation. A transfer rate measurement algorithm is implemented in the receiver element. This algorithm counts the received user data payloads from the two sensor elements. The amount of received user data payload per second is used to calculate the transfer rate. The latencies caused by the ADC are estimates based on datasheets from the microcontroller manufacturer. The serial communication latencies are measured by a MAX/MSP patch which counts receiving frames from the serial object. These measurements and estimates are shown in Chapter 4.



(a) Main routine (b) Send ADC routine (c) Timer interrupt routine (d) ADC interrupt routine

Figure 3.7: Flowchart for the sensor element

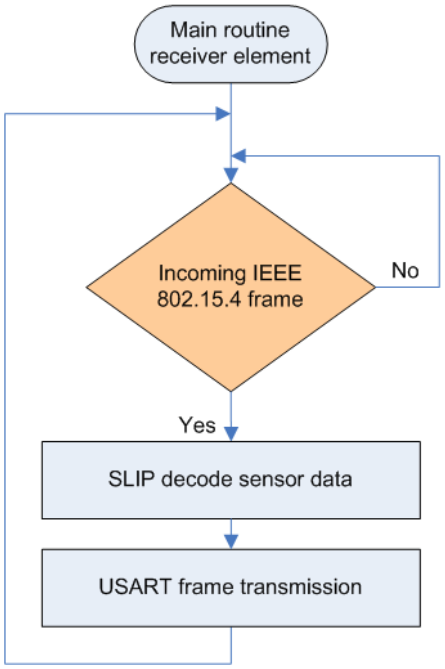


Figure 3.8: Flowchart for the receiver element

3.2.3 Program flow

Figure 3.7 shows the flowcharts for the program implemented in the sensor element's microcontroller. It shows the program flow from the ADC measurement to the wireless transmission of the ADC data. The blue boxes are implemented in this thesis while the yellow box shows the IEEE 802.15.4 frame transmission implemented in the AVR2001 software provided by Atmel.

The program start in the main routine, see Figure 3.7a. The main routine handles the initializations and starts the send routine, see Figure 3.7b. By setting a flag[4] at reasonable intervals, a timer is used to control the interval between IEEE 802.15.4 transmissions. The interval between these transmissions has to be greater than the time needed to perform the ADC measurements. Figure 3.7d shows the ADC interrupt routine. The send routine, see Figure 3.7b, awaits the flag set by the Timer interrupt, and starts the frame transmission when this flag occurs.

Figure3.8 shows the program flow for the receiver element's microcontroller. The incoming IEEE 820.15.4 frames contains ADC data which is SLIP decoded(see Section 2.4) and retransmitted as USART frames into MAX/MSP.

When transmitting frames, the AVR2001 software uses the extended radio transceiver states (see Section 2.10.1). These states enable IEEE 802.15.4 features like CSMA/CA (see Section 2.3.1). A flowchart for the IEEE 802.15.4 frame transmission routine is given in the appendix.

3.3 A star motion capture system based on the IEEE 802.15.4 standard

This section describes the implementation of the star motion capture system based on the IEEE 802.15.4 standard (see Section 2.3 and 3.1.2). This solution is desirable if a complete wireless setup is needed. As opposed to a peer-to-peer network, each node doesn't need to measure a high number of ADC channels. Therfore, the latency caused by the ADC will be less than in a peer-to-peer network. To avoid collisions and failed transfers, acknowledgements, Cyclic Redundancy Check (CRC) and other security methods are useful.

Overall system latency tests is described in Chapter 4. This will point out how suitable this setup will be in an *Active Music* setting, how many sensor elements it's reasonable to include for one receiver element and how much processing it's reasonable to perform inside each sensor element.

3.3.1 AVR2002

AVR2002 - AT86RF230 Raven Evaluation Software is an application note which is designed to evaluate the AT86RF230 radio transceiver using the AVRAVEN device (see Section 2.11). It provides codes for various tests [5] that users can exploit and modify.

The AVR2002 is built upon the Atmel MAC software stack (Section 2.12.2), while the TAT (Section 2.12.1) and the HAL layer (Section 2.3.1) provide the functions needed to control

[4] This flag can be a variable that is set each time an interrupt occurs and can be used to start and stop other part of the program.

[5] A packet error rate/range characterization test, a RF characterization test and a DC characterization test

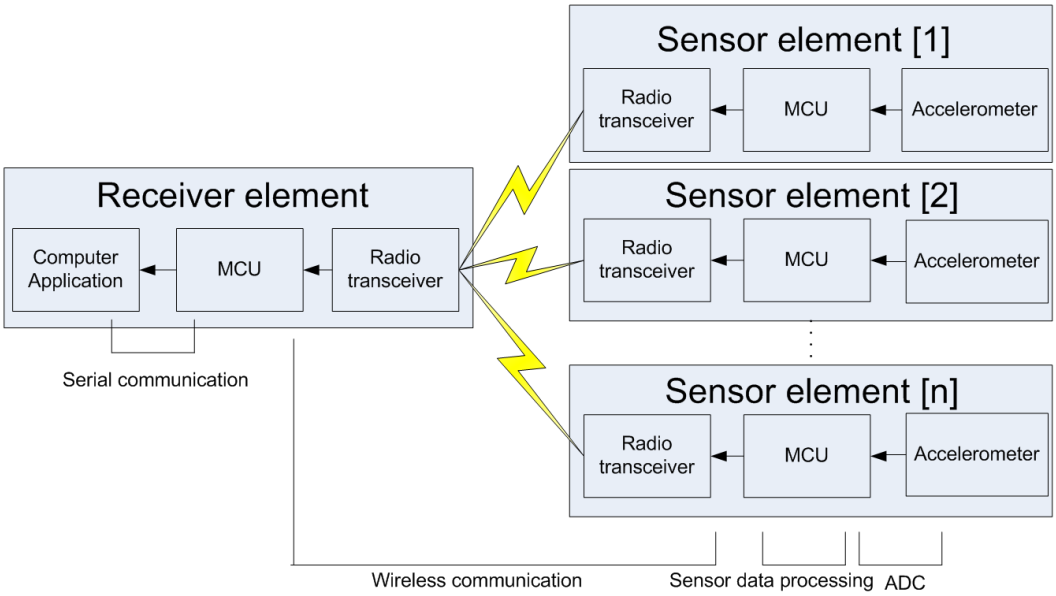


Figure 3.9: Possible latency sources in the AVR 2002 star example

the radio transceiver. The RZUSBSTICK acts as a USB to serial converter [6] and an USART communication (see Section 2.6.3) can be used in the serial transmission.

The AVR2002's DC characterization test provides a good starting point for a star based motion capture system. This test enables the user to measure current consumption over time when multiple AVRaven boards transmit data to the RZUSBSTICK.

By removing part of the featured AVR2002 code while implementing features like ADC measurements, the AVRaven boards can act as sensor elements while the RZUSBSTICK can, along with a computer, act as a receiver element.

IEEE 802.15.4 features like CSMA/CA, wasn't implemented in the AVR2002 featured codes. Further implementations of these features was difficult because of the complexity of the IEEE 802.15.4 standard. A solution to avoid collisions is to use different channels for each sensor element. This means that each sensor element transmits on different channels while the receivers element change channel after each frame reception. This is a solution that will decrease the possible transfer rate. Therefore a theoretical calculation for a star based motion capture network with CSMA/CA and acknowledgement will be presented in Chapter 4. This will be used to compare this implementation with a possible improvement.

3.3.2 Latency sources

Latency sources in this star motion capture system setup is shown in Figure 3.9. These latency sources are the same as those described in Section 3.2.2.

The ADC conversion latency will be smaller in this setup compared to the peer-to-peer setup because of the amount of accelerometers per microcontroller. The latencies caused by IEEE 802.15.4 transmission will be increased because of the channel change implementation.

[6]The Communication Device Class (CDC) driver software enables the USB connection to appear as a virtual COM port

Different latency tests are conducted on this implementation. A transfer rate measurement algorithm is implemented in the receiver element. This algorithm counts the received user data payloads from the two sensor elements. The amount of received user data payload per second is used to calculate the transfer rate. The latencies caused by the ADC are estimates based on datasheets from the microcontroller manufacturer. The serial communications latencies are measured by a MAX/MSP patch which counts receiving frames from the serial object. These measurements and estimates are shown in Chapter 4.

3.3.3 Program flow

Figure 3.10 shows flowcharts for accelerometer reading and transmission through a star motion capture system. The blue boxes are implemented in this thesis while the yellow box shows the IEEE 802.15.4 frame transmission implemented in the AVR2002 software provided by Atmel.

The RZUSBSTICK creates SLIP frames (see Section 2.4) of the received sensor data. The SLIP encapsulation simplifies monitoring and storing in computer applications. The RZUSBSTICK receives data from multiple sensor elements. After each reception a sniffer mode feature in the RZUSBSTICK change the operation channel.

Figure 3.10a show a high level flowchart of the sniffer program realized in the RZUSBSTICK. After each reception, a Res Report Sniffer routine is called. This routine change the channel, create SLIP frames with the received sensor data and transmit these frames through an USART connection.

Figure 3.10b shows the Res Frame Payload routine which is realized in each sensor element. This routine is called as often as possible. It starts the ADC conversation, waits for the preferred amount of ADC data and then transmit these data in IEEE 802.15.4 frames.

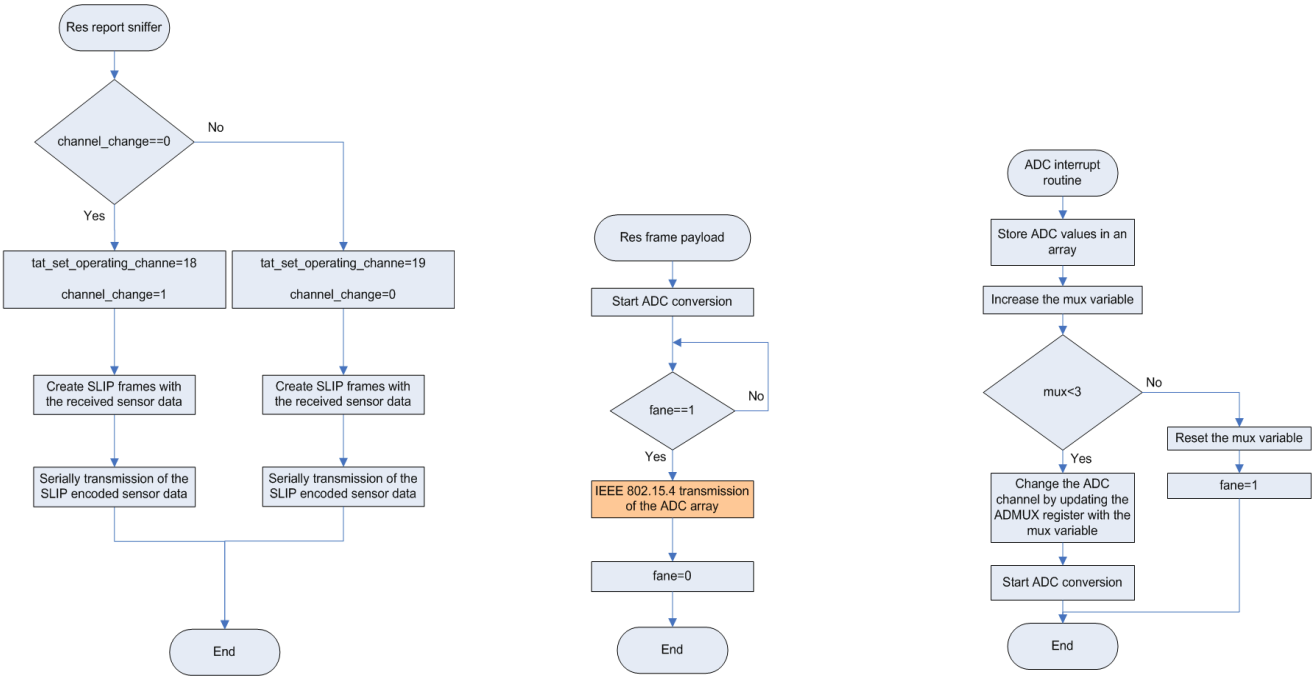
Figure 3.10c shows the ADC interrupt routing which is called by the Res Frame Payload routine. This routine generates 10 bit digital values based on the measured analog output voltages from the accelerometers. After each conversion, the ADC channel is changed by increasing the ADMUX register. An accelerometer outputs 3 values. Therefore, the ADC conversion interrupt is reset after the third conversion and a variable is set. This variable triggers an IEEE 802.15.4 transmission in the Res Frame Payload routine.

When transmitting frames, this example only uses the basic radio transceiver states (see Section 2.10.1). A flowchart for the IEEE 802.15.4 frame transmission routine is given in the appendix.

3.3.4 MAX/MSP monitoring

This implementation includes a monitoring application realized in MAX/MSP. Figure 3.11 shows a patch which reads data from the serial port, SLIP decodes the data and displays the data in a graph. The graphs shows accelerometer data from the two sensor elements in the implemented star based motion capture system.

The ADC measurements result in 10 bit digital values. The 10 bit values are separated into two 8 bit values before transmission from the sensor element to the receiver element. The sel object in the MAX/MSP patch adds the 8 bit values and outputs values between 0 and 1023. These values are monitored in the multislider object.



(a) The USB sniffer (b) The IEEE 802.15.4 payload generator (c) The ADC interrupt routine

Figure 3.10: Flowcharts for the star based wireless motion capture network based on the AVR 2002 application note

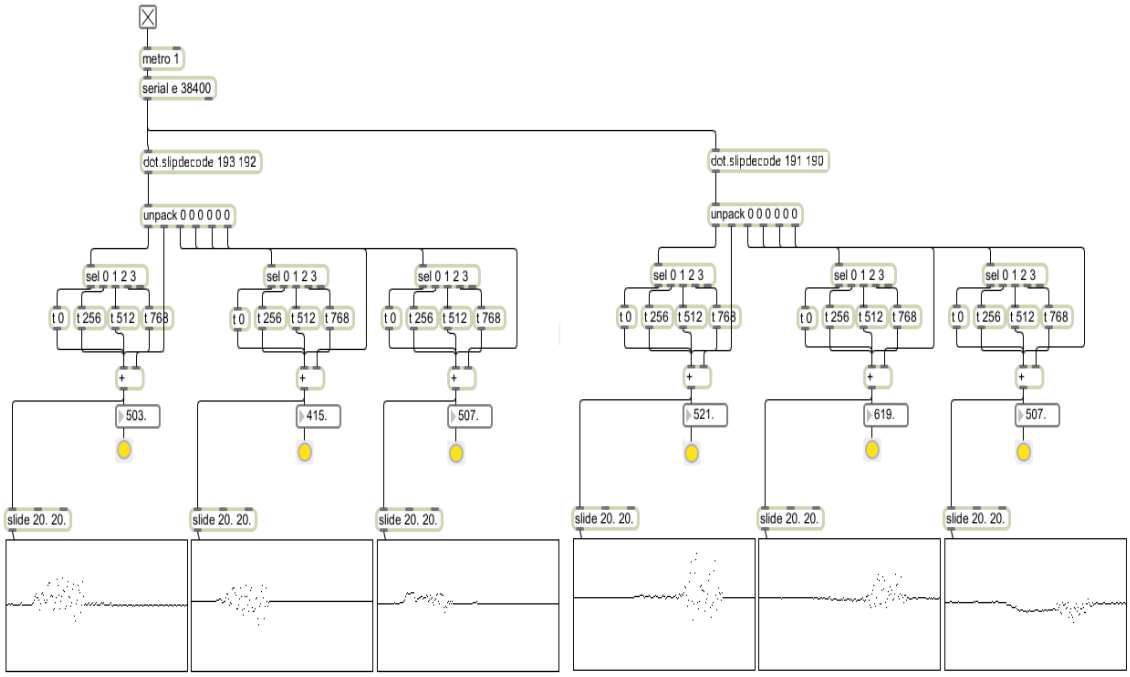


Figure 3.11: A MAX/MSP patch for monitoring sensor data from multiple sensor nodes

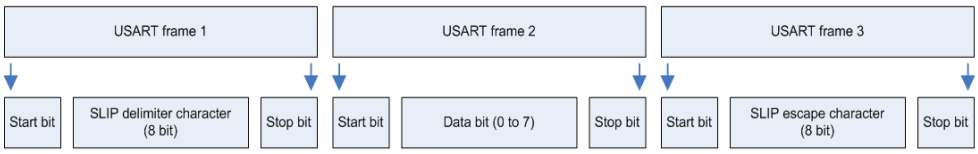


Figure 3.12: One SLIP decoded single Byte

3.4 Serial data receiving

This section describes the solution for communication within the receiver elements. The reseiver element's microcontroller has to be able to transmit data into the receiver element's computer. The implemented motion capture systems use a serial connection for this communication.

This communication is based on the USART (see Section 2.6.3) standard. This section shows sizes of USART frames, when they are consisting of different amount of accelerometer data. This information is helpful for understanding the latencies measurement in Chapter 4.

The USART data frame will be a binary string containing values from accelerometers. In the peer-to-peer based motion capture system, these USART frames will consist of data from multiple accelerometers[7]. In the star based motion capture system, these USART frames will consist of data from one accelerometer[8].

A secure USART transmission within the receiver element requires an additional protocol. The implementations in this thesis use the SLIP protocol (see Section 2.4). The baud rate gives the possible transfer rate in a USART transmission. This transfer rate will be reduced because of USART start and stop bits and SLIP headers and tails. The following equation gives the size of an USART frame that consists of data from one accelerometer (6 bytes):

$$\begin{aligned} \text{USART frame size[bits]} &= \text{Accelerometer data[bytes]} \\ &+ \text{SLIP headers and tails[bytes]} \\ &+ \text{USART start and stop bits[bits]} \end{aligned}$$

$$\begin{aligned} \text{USART frame size[bits]} &= 6 \text{ byte} \\ &+ 7 \text{ Byte} \\ &+ 26 \text{ bits} \\ &= 130 bits \end{aligned}$$

An example is given in Figure 3.12. This is a USART transmission of one SLIP decoded byte. Table 3.1 shows USART frame sizes for different number of ADC channels. USART transfer rate and latency measurements are conducted in Chapter 4.

[7]The sensor element transmit one frame which consist of data from multiple accelerometers. The receiver element retransmits all accelerometer data in one USART transmission

[8]The sensor elements transmit frames which consist of data from single accelerometer. The receiver element retransmits accelerometer data from one sensor element at a time

	10-bit
1 accelerometer (6 Bytes)	130 bits
2 accelerometer (12 Bytes)	250 bits
3 accelerometer (18 Bytes)	370 bits
4 accelerometer (24 Bytes)	490 bits

Table 3.1: UART frame sizes for SLIP decoded ADC values at different resolution

3.5 Sensor data processing within the sensor element

The microcontroller can be used as a simple sensor data receiver which measure sensor data and retransmit these data's through the wireless link. This way the sensor elements acts as a router, where the received sensor data is retransmitted as fast as possible.

In addition to acting as a sensor data router, the microcontroller can perform different filter algorithms or extract important information from the sensor data. These kind of algorithms can be used to minimize the amount of sensor data to be sent, while decrease the amount of important information within these data.

3.5.1 Filter implementations

During this thesis, some simple filters were implemented in the sensor elements. A median filter, a mean filter and a high pass filter (see Section 2.13). This was done to estimate on the possible latency which sensor element processing will apply. The same filters were implemented in MAX/MSP. The monitored result from both sensor element processing and receiver element processing is shown in Chapter 4.

The processing algorithm starts with a median filter, then a mean filter and finally a high pass filter. The matrixes below show the steps from raw to filtrated sensor data, when using median, mean and high pass filter. After an array is filled with sensor data, the program calculates the median values of each row in the array.

$$\begin{bmatrix} X_1 & X_2 & \cdots & X_n \\ Y_1 & Y_2 & \cdots & Y_n \\ Z_1 & Z_2 & \cdots & Z_n \end{bmatrix} = \begin{bmatrix} X_{median} \\ Y_{median} \\ Z_{median} \end{bmatrix}$$

After another array is filled with median values, the program calculates the mean values of each row in the array. A high pass filter then excludes low frequencies. The output from the high pass filter is sent into MAX/MSP.

$$\begin{bmatrix} X_{median_1} & X_{median_2} & \cdots & X_{median_n} \\ Y_{median_1} & Y_{median_2} & \cdots & Y_{median_n} \\ Z_{median_1} & Z_{median_2} & \cdots & Z_{median_n} \end{bmatrix} = \begin{bmatrix} X_{mean} \\ Y_{mean} \\ Z_{mean} \end{bmatrix} = \begin{bmatrix} X_{filtrated} \\ Y_{filtrated} \\ Z_{filtrated} \end{bmatrix}$$

Flowchart

Figure 3.13 shows a flowchart for the microcontroller program. The ADC outputs 3 values, one for each axis on a triaxial accelerometer. The first decision box awaits a signal from the ADC

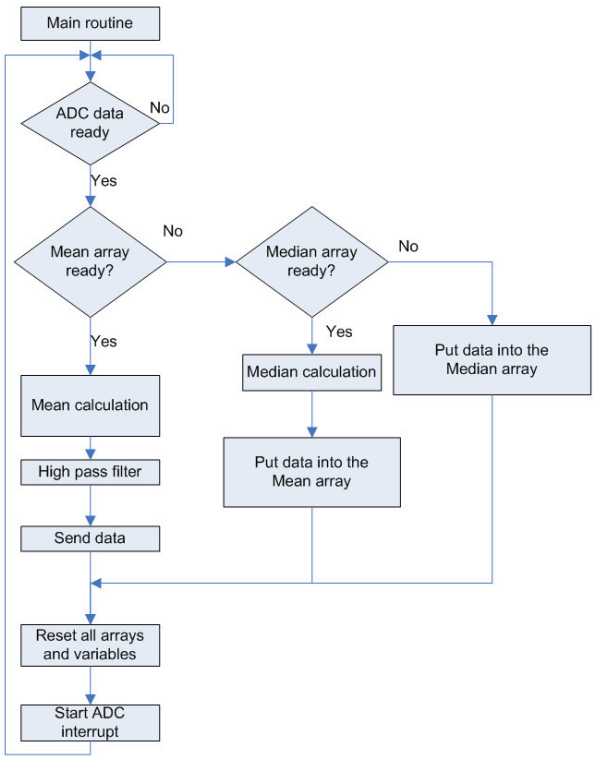


Figure 3.13: Flowchart for filter implementation in the sensor element

interrupt which tells if 3 ADC values are converted. The converted ADC values are stored in an array for median calculation. The median filter removes spikes in the signal, so it's reasonable to start with this filter. This is repeated until the median array is filled with data and the median calculation can start. The median filter output 3 values, one for each axis, which is stored in an array for mean calculation. For each new value in the mean array, a new median calculation has to be performed. When the mean array is filled, the microcontroller calculates the mean value. The mean filter output 3 values, one for each axis, which goes through a high pass filter before transmission.

Figure 3.14 shows a high level presentation of a median filter implemented in the microcontroller. A bobble sort algorithm adjusts the order of a string of numbers, so the numbers will appear in an increasing order. The median value is the midmost number in the string after a bobble sort. This is the most time consuming of the implemented filters. This will be considered as a reference for the possible amount of sensor processing which is reasonable to implement in the sensor element. Measurement in Chapter 4 will show the time needed for median calculations in the sensor elements.

MAX/MSP patch

The same filters, as described above, are implemented in a MAX/MSP patch. Figure 3.15a shows a MAX/MSP patch for monitoring the filtrated data from the sensor element. Figure 3.15b shows a MAX/MSP patch which receive raw sensor data from the sensor elements, and perform the additional filtrating. This patch later used to compare receiver element processing and sensor element processing.

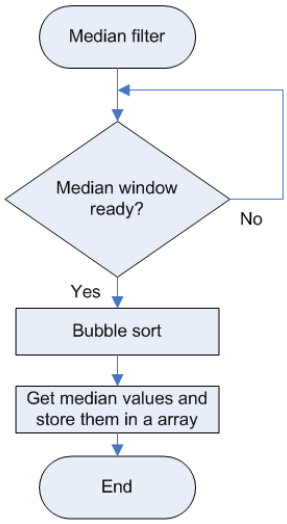
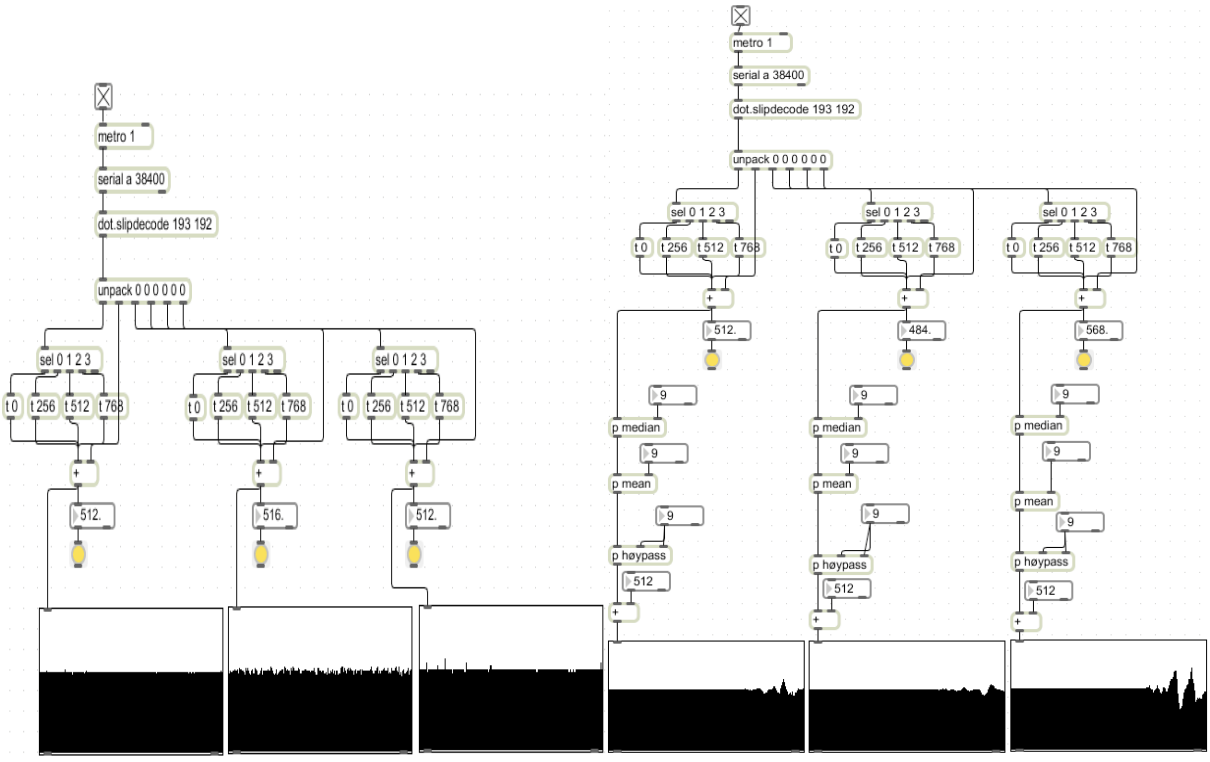


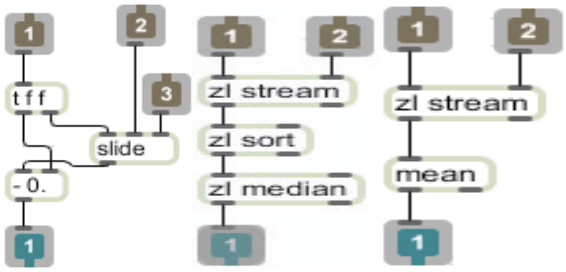
Figure 3.14: High level presentation of a hardware median calculation



(a) A MAX/MSP patch monitoring filtrated sensor data from the sensor elements

(b) A MAX/MSP patch monitoring and filtrating raw sensor data from the sensor elements

Figure 3.15: Implemented MAX/MSP patches



(a) A MAX/MSP high-pass filter (b) A MAX/MSP median filter (c) A MAX/MSP mean filter

Figure 3.16: MAX/MSP filters

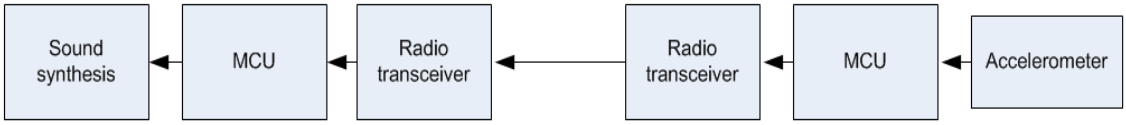


Figure 3.17: Drum setup

Figure 3.16a, 3.16b and 3.16c shows the filters which is used in the MAX/MSP patch.

3.6 An *Active Music* Application

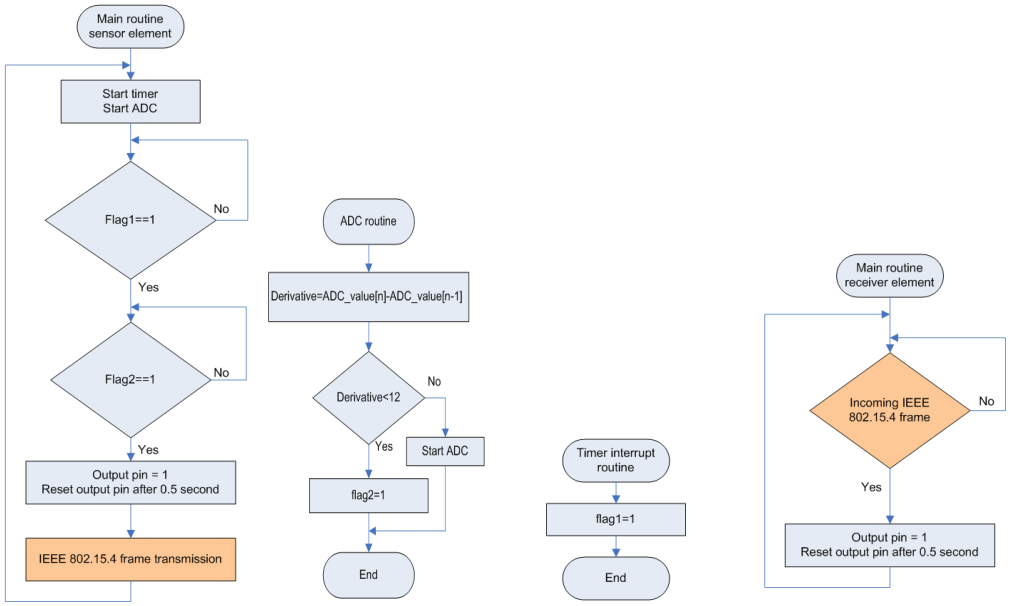
A wireless motion based drum synthesis is a possible *Active Music* application that can be based on the implemented motion capture systems in this thesis. A person is pretending to hit drums without any actual drums or drumsticks. An accelerometer measures motions while a sound synthesis generates a suitable sound whenever the nature of the movements approaches the movement of a typical drum stroke.

This section describes an implementation, done in this thesis, that can be a part of this application. This implementation is based on the peer-to-peer motion capture system in Section 3.2.

A sensor element measures the x axis of an accelerometer and calculates the derivative value. The derivative value is compared to a threshold. Whenever this threshold is reached, a signal is sent through the wireless link and into a sound synthesis in the receiver element. This implementation will show the latencies from the ADC measurements and across the wireless link. Further transmission into a sound generating application will not be tested in this implementation.

This implementation uses an 8-bit resolution. Therefore the ADC clock can be set as high as 1000KHz (Section 4.1.1). Only one of the accelerometers axis needs to be converted, so the overall conversion time will be 13 μs.

Results from this test is shown in Chapter 4.



(a) Transmitters main routine (b) Send ADC routine (c) Timer interrupt routine (d) Receivers main routine

Figure 3.18: Flowchart for the *Active Music* application

3.6.1 Flowcharts

Figure 3.18 shows the flowcharts for this implementation. The blue boxes are implemented in this thesis while the yellow box shows the IEEE 802.15.4 frame transmission implemented in the AVR2001 software provided by Atmel.

The transmitters main routine (Figure 3.18a) starts the Timer and the ADC, and awaits two flags. These flags are set after a timer interrupt (Figure 3.18c) and when data from the ADC is ready.

The ADC interrupt (Figure 3.18b) converts one value from the accelerometer and calculate the derivative (Section 2.13), which is the difference between the present value and the previous ADC value. The derivative, which is zero when the accelerometer isn't moving, is compared to a threshold. If this threshold is reached, the flag inside the ADC interrupt is set.

After both flag is set, the transmitters main routine sets and output pin and starts the IEEE 802.15.4 transmission. The receiver's main routine (Figure 3.18d) awaits the IEEE 802.15.4 frame and then sets an output pin.

An oscilloscope can be used to measure the latency in this system. The output pins on the transmitter and the receiver can be connected to separate channels. And then the oscilloscope can show the latency between these two signals.

Chapter 4

Results

The following experiments are based on the implementations outlined in Chapter 3. This chapter describes the quality of these implementations according to latencies and transfer rates.

The following table summarizes the tests done in this chapter.

Tests	Methods	Section
ADC latency	Latency estimates	4.1
Serial Transmission in the receiver element	Latency measurements and estimates	4.2
A peer-to-peer motion capture system	Latency measurements and estimates	4.3
A star motion capture system	Latency measurements and estimates	4.4
Sensor data processing in the sensor element	Latency measurements and estimates	4.7
A drum application	Latency measurements	4.6

The following results will be compared to a latency requirement for musical applications proposed in [17], which sets the acceptable upper bound for a sound synthesis' audible reaction to motions at 10 ms.

4.1 ADC latency

The ADC in the sensor element is a possible latency source in the system. This section describes the estimated latencies in a motion capture system caused by the ADC conversions (see Section 2.6.2). These estimates are based on the ADC conversion times given in Atmega1281V's datasheet [6]. The ADC latencies, obtained in this section, will be used in the following sections in order to get an overview of the total latencies in the motion capture systems.

4.1.1 ADC conversion time

The microcontrollers used in this thesis feature 10-bit ADCs. The Atmega1281v has 8/16 single-ended ADC channels (the 100 pin version offers 16 channels and the 64 pin version offers 8 channels) and the Atmega1284P has 8 single ended ADC channels. The implemented

Number of ADC inputs	ADC clock frequency				
	50 KHz	**125 KHz**	**200 KHz**	**250 KHz**	**1000 KHz**
1	260 μs	104 μs	65 μs	52 μs	13 μs
2	520 μs	208 μs	130 μs	104 μs	26 μs
3	780 μs	312 μs	195 μs	156 μs	39 μs
4	1040 μs	416 μs	260 μs	208 μs	52 μs
5	1300 μs	520 μs	325 μs	260 μs	65 μs
6	1560 μs	624 μs	390 μs	312 μs	78 μs
7	1820 μs	728 μs	455 μs	364 μs	91 μs
8	2080 μs	832 μs	520 μs	416 μs	104 μs
9	2340 μs	936 μs	585 μs	468 μs	117 μs
10	2600 μs	1040 μs	650 μs	520 μs	130 μs
11	2860 μs	1144 μs	715 μs	572 μs	143 μs
12	3120 μs	1248 μs	780 μs	624 μs	156 μs
13	3380 μs	1352 μs	845 μs	676 μs	169 μs
14	3640 μs	1456 μs	910 μs	728 μs	182 μs
15	3900 μs	1560 μs	975 μs	780 μs	195 μs
16	4160 μs	1664 μs	1040 μs	832 μs	208 μs

Table 4.1: ADC conversion time for multiple inputs.

peer-to-peer based motion capture system uses the 64 pin version ATmega1281v microcontroller in sensor element and the star based motion capture system uses the Atmega1284P microcontroller in the sensor element. The ADC conversion time on both microcontrollers is between 13 μs and 260 μs given by the prescaler settings (see Section 2.6.2). A single conversion takes 13 ADC clock cycles. The following equation gives the ADC conversion time [6]:

$$ADC_{Conversion\_time} = \frac{1}{ADC_{clock}} * 13cycles \tag{4.1}$$

The prescaler selection bits offer different prescaler division factors. These are used to adjust the ADC clock frequency. The recommended ADC clock frequency is between 50 KHz and 200 KHz when using 10 bit resolution. The ADC clock frequency can be as high as 1000 KHz when using 8 bit resolution. In order to get an accurate ADC conversion when using 10 bit resolution, the division factor has to be 64. This results in a 125 KHz ADC clock frequency which is used in all implementations in this thesis except from the Drum Application. The Drum Application uses an 8 bit resolution that allows a 1000 KHz ADC clock frequency.

The 100 pin ATmega1281v contains 16 single ended ADC channels. This means it can be used to read 5 accelerometers. The implementation in this thesis uses the 64 pin ATmega1281v which can only measure 2 accelerometers. The reason for this choice is that the 100 pin ATmega1281v needed an additional connection kit. These two microcontrollers has the same characteristic, therefore 5 accelerometers will be included in the latency calculations.

The following tables show the time needed for conversions. Table 4.1 shows the ADC conversion time when using different numbers of ADC inputs and Table 4.2 shows the ADC conversion time when reading multiple ADXL330 3-axis accelerometers.

Number of 3-axis accelerometers	ADC clock frequency				
	50 KHz	**125 KHz**	**200 KHz**	**250 KHz**	**1000 KHz**
1	780 μs	312 μs	195 μs	156 μs	39 μs
2	1560 μs	624 μs	390 μs	312 μs	78 μs
3	2340 μs	936 μs	585 μs	468 μs	117 μs
4	3120 μs	1248 μs	780 μs	264 μs	156 μs
5	3900 μs	1560 μs	975 μs	780 μs	195 μs

Table 4.2: ADC conversion time when reading multiple accelerometers.

	8-bit		**10-bit**	
	1 accelerometer (3 SLIP decoded ADC values)	**2 accelerometer (6 SLIP decoded ADC values)**	**1 accelerometer (3 SLIP decoded ADC values)**	**2 accelerometer (6 SLIP decoded ADC values)**
9600 bps	123 fps	66 fps	66 fps	35 fps
19200 bps	246 fps	133 fps	133 fps	69 fps
38400 bps	492 fps	264 fps	264 fps	139 fps
76400 bps	983 fps	530 fps	530 fps	275 fps

Table 4.3: Received frames per second (fps) for different frame sizes, different resolutions and different baud-rate settings.

4.1.2 Conclusion

The ADC latencies will affect the implemented motion capture systems in different ways. It is obvious that processing algorithms which demand several series of ADC data, such as median filter and mean filter, will have an increasing latency when reading and processing datas from multiple accelerometers. The peer-to-peer based motion capture system sensor will therefore be influenced by these latencies to a higher degree than the star based motion capture system.

4.2 Serial Transmission

The motion capture systems implemented in Chapter 3 uses serial transmission for hardware/-software communication within the receiver elements (see Section 3.4). This section describes the measured latencies caused by this communication.

A serial communication is not a desirable solution compared to other communication protocols with higher transfer rates such as USB. Overall latency measurements and estimates in the following section will therefore be presented with and without the latencies caused by the serial transmission.

4.2.1 Serial transmission within the receiver element

Section 3.4 describes how accelerometer data can be SLIP decoded and built into USART frames. This section presents measurements that describe the suitability of a serial USART

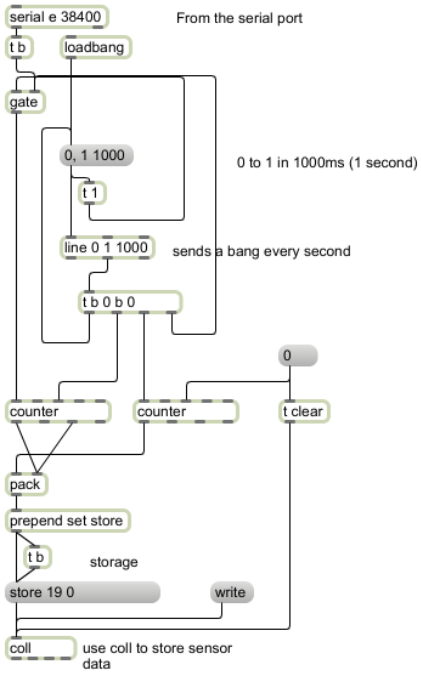


Figure 4.1: A MAX patch measuring received frames per second from the serial object

transmission within the receiver elements. This USART transmission includes SLIP decoded accelerometer data.

MAX/MSP offers a serial object that enables serial data to be transmitted into MAX/MSP. MAX/MSP allows a 76400 bps baud rate. Received accelerometer data per seconds is measured by a patch shown in Figure 4.1. This patch is taken from [16]. One frame in this example means one set of accelerometer data. This patch is used for measurement at different baud rates and different amount of accelerometer data. Table 4.3 shows the number of received frames per second from the USART communication. These frames include SLIP characters and accelerometer data.

The following equation shows the interval between each received set of SLIP decoded accelerometer data, when data from two accelerometers is SLIP decoded and received by MAX/MSP 275 times each second.

$$\frac{1}{275fps} = 3.64ms \tag{4.2}$$

The USART transmission will therefore result in 3.64 ms latency.

4.3 A peer-to-peer motion capture system

This section describes the latency measurements done on the peer-to-peer motion capture system implemented in Section 3.2. Effective transfer rates[1] on this system have been measured during this thesis. This measurement can be used to obtain the latency across the wireless link when transmitting different amount of accelerometer data.

[1]The user data transfer rate

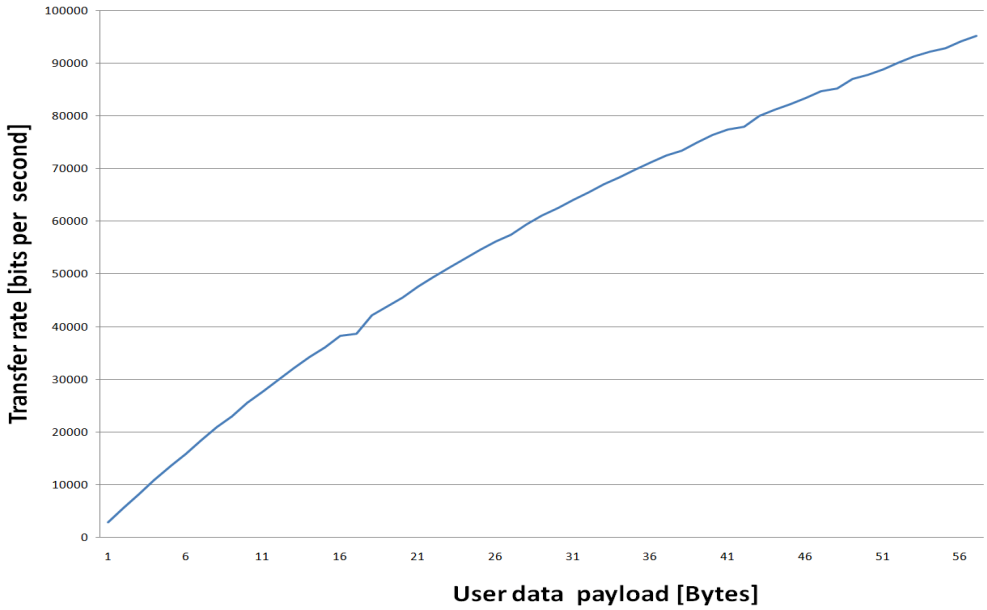


Figure 4.2: The measured IEEE 802.15.4 transfer rates as a function of user data payload

4.3.1 IEEE 802.15.4 limitations

The measurement in Figure 4.2 shows how the amount of user data payload influences the transfer rate. The x-axis shows the amount of user data payload in each IEEE 802.15.4 frame. The y-axis shows the amount of bits received by the receiver element per second. The measurement is done with a distance of two meters between the sensor element and the receiver element.

The payload needed for this sensor application will probably be between 6 bytes and 32 bytes according to how many accelerometers which is measured by the sensor elements. The transfer rate will therefore be between 15Kbps and 62Kbps (see Figure 4.2).

The following formula shows how to calculate the interval between each frame reception based on the measured transfer rate:

$$\text{Interval between received frames} = \frac{\text{User data payload [bits]}}{\text{Transfer rate [bits per second]}} \tag{4.3}$$

Figure 4.3 shows the interval between received frames at different user data payload. This shows how the latency between each IEEE 802.15.4 frame increases according to the user data payload.

4.3.2 Conclusion

Table 4.4 shows the total latencies in this system when transmitting 10-bit ADC data. Further use of sensor data, such as motion recognition, will require high resolutions. This solution offer secure transmissions by using IEEE 802.15.4 features like Acknowledgements and CSMA/CA.

Table 4.5 shows the latency from the accelerometers into MAX/MSP. This shows how the USART transmission will influence the latency and make this system unsuitable for *Active Music*. The latency approaches the *Active Music* latency requirement after 3 accelerometers.

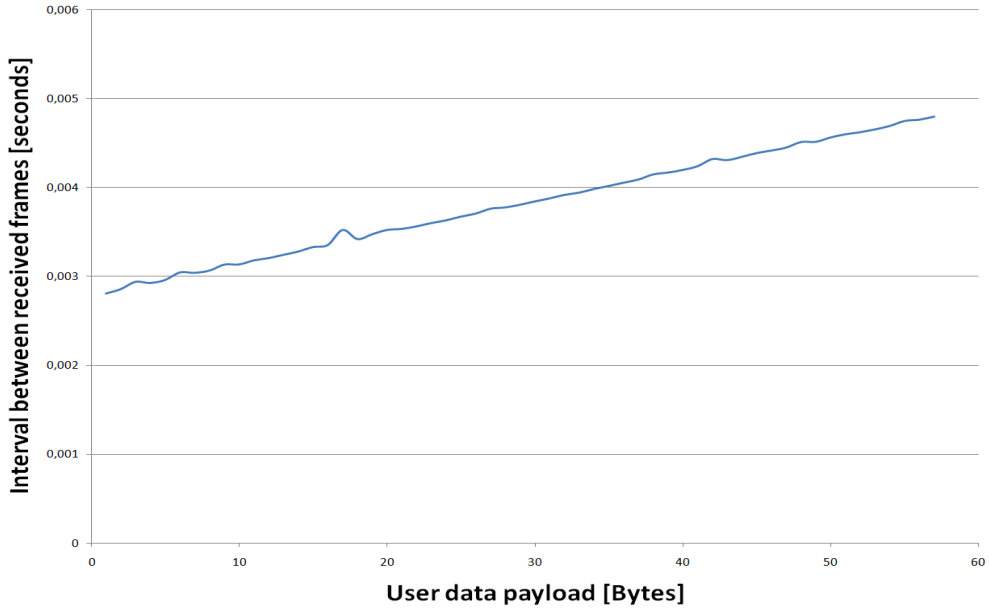


Figure 4.3: The measured interval between received frames as a function of user data payload

	ADC conversion time	**IEEE 802.15.4 latency**	**Total latency**
1 accelerometer (3 bytes)	312 μs	3.04 ms	3.352 ms
2 accelerometer (6 bytes)	624 μs	3.20 ms	3.824 ms
3 accelerometer (9 bytes)	936 μs	3.42 ms	4.356 ms
4 accelerometer (12 bytes)	1248 μs	3.62 ms	4.4868 ms
5 accelerometer (15 bytes)	1560 μs	3.84 ms	5.4 ms

Table 4.4: The total latency of the hardware system

	ADC conversion time	IEEE 802.15.4 latency	Serial SLIP transmission at 76800 bps	Total latency
1 accelerometer (3 bytes)	312 μs	3.04 ms	1.89 ms	5.24 ms
2 accelerometer (3 bytes)	624 μs	3.20 ms	3.64 ms	7.46 ms
3 accelerometer (9 bytes)	936 μs	3.42 ms	5.4 ms (estimated)	9.76 ms
4 accelerometer (12 bytes)	1.25 ms	3.62 ms	7.14 ms (estimated)	12.01 ms
5 accelerometer (15 bytes)	1.56 ms	3.84 ms	8.8 ms (estimated)	14.2 ms

Table 4.5: The total latency from accelerometer to MAX/MSP

The possible time which can be used to process sensor data is decreased while adding accelerometers. Several series of multiple accelerometer measurements will not be possible within 10 ms. A limited amount of sensor processing will therefore be possible carry out in a peer-to-peer based motion capture system consisting of 5 accelerometers.

4.4 A star motion capture system

This section describes the latency measurements done in the peer-to-peer motion capture system implemented in Section 3.2. The measurement is carried out with two sensor elements communicating with one receiver element. Effective transfer rates are measured, which can show the latencies across the wireless link.

4.4.1 Multiple sensor elements in a star network

Figure 4.4 shows the measured transfer rates as a function of user data payload when using 2 sensor elements. Figure 4.5 shows the interval between received frames as a function of different user data payloads.

An additional measurement with 1 sensor element were done in order to see how much the channel change interferred with the transfer rates. Figure 4.6 shows how the transfer rate decreases when using 2 sensor elements compared to 1 sensor element. The green line shows the transfer rate when the sniffer receives data from 1 sensor element. The yellow and blue line show the transfer rate when the sniffer receives data from 2 sensor elements. This graph shows a 50 percent transfer rate reduction when using 2 sensor elements instead of 1.

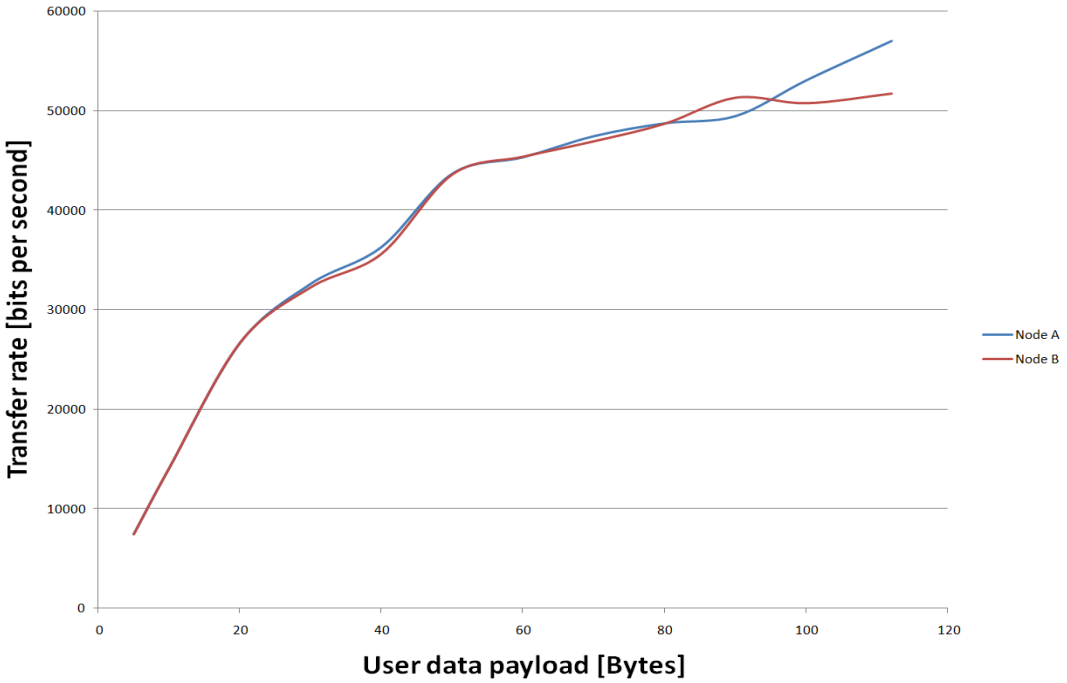


Figure 4.4: Measured transfer rates as a function of the user data payload when using 2 sensor elements

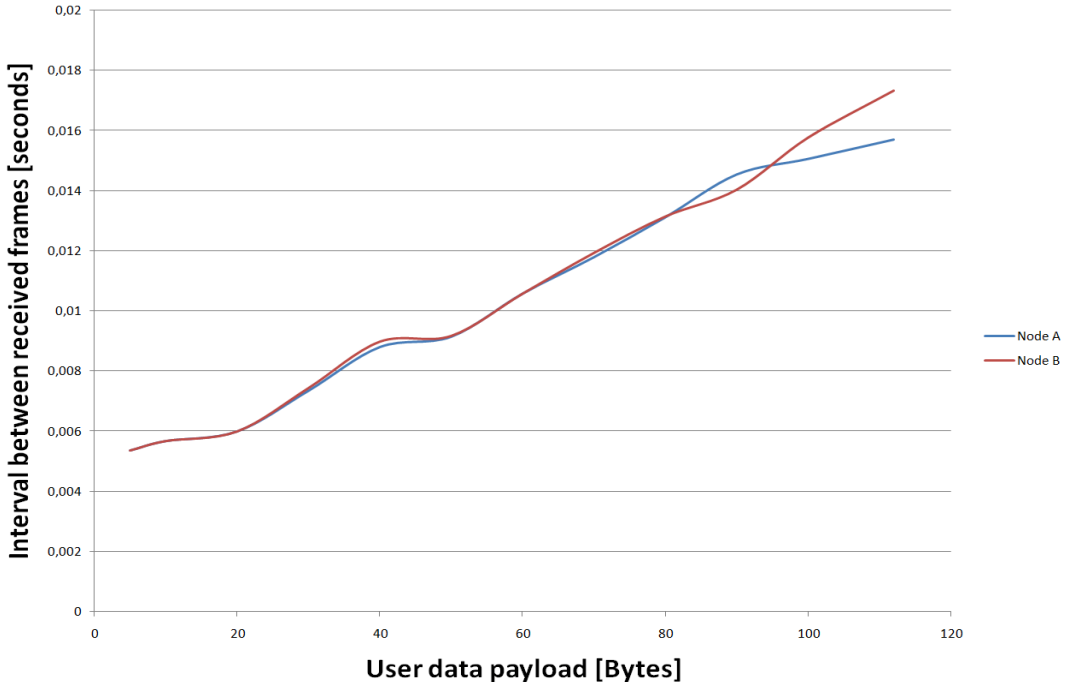


Figure 4.5: Measured interval between received frames as a function of different user data payload when using 2 sensor elements

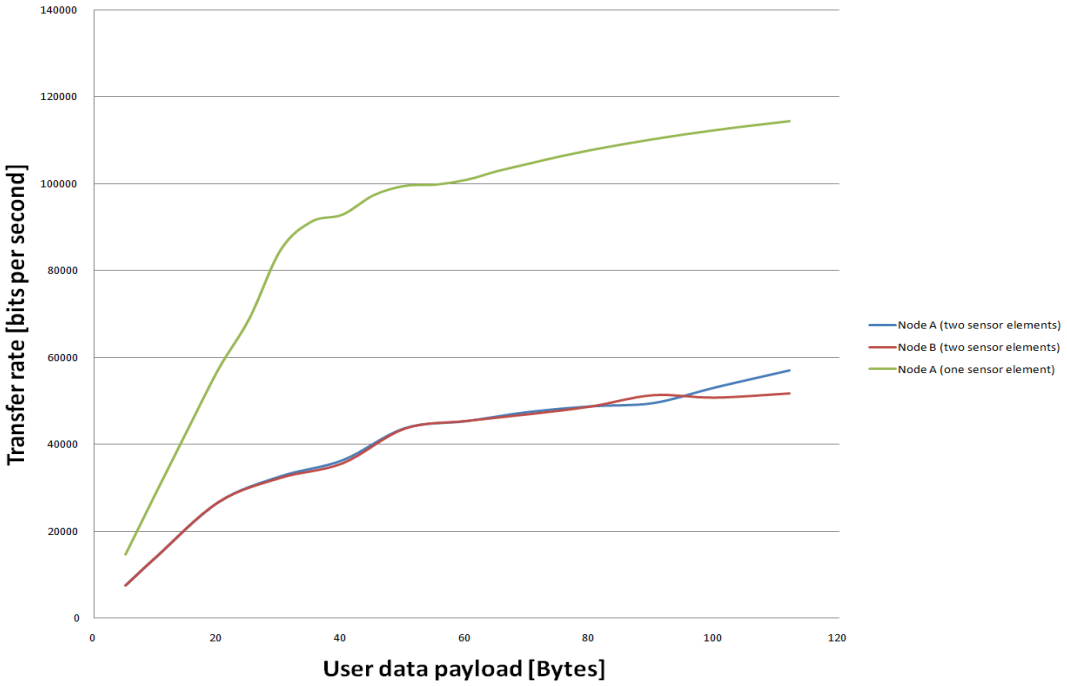


Figure 4.6: Compared transfer rates

Conclusion

An additional estimation based on the transfer rate reduction when using 2 sensor elements instead of 1, gives the possible transfer rate with 3 and 4 sensor elements. Table 4.6 shows the latencies including ADC conversions and IEEE 802.15.4 transmission. Table 4.7 shows the system latencies including ADC conversions, IEEE 802.15.4 transmission and serial USART transmission. This setup will in an *Active Music* application have a limitation of 3 sensor elements when excluding the latency caused by the serial transmission.

	ADC conversion time	**IEEE 802.15.4 latency**	**Total latency**
1 sensor element	312 μs	2.729 ms	3.041 ms
2 sensor element	312 μs	5.232 ms	5.544 ms
3 sensor element	312 μs	8.18 ms(estimated)	6.29 ms
4 sensor element	312 μs	10.03 ms(estimated)	10.342 ms

Table 4.6: The total latency of the hardware system with multiple sensor elements in a star based setup

	ADC conversion time	IEEE 802.15.4 latency	Serial SLIP transmission at 76800 bps	Total latency
1 sensor element	312 μs	2.729 ms	1.89 ms	4.931 ms
2 sensor element	312 μs	5.232 ms	1.89 ms	7.434 ms
3 sensor element	312 μs	8.18 ms(estimated)	1.89 ms	10.382 ms
4 sensor element	312 μs	10.03 ms(estimated)	1.89 ms	12.232 ms

Table 4.7: The total latency of the hardware and software system with multiple sensor elements in a star based setup

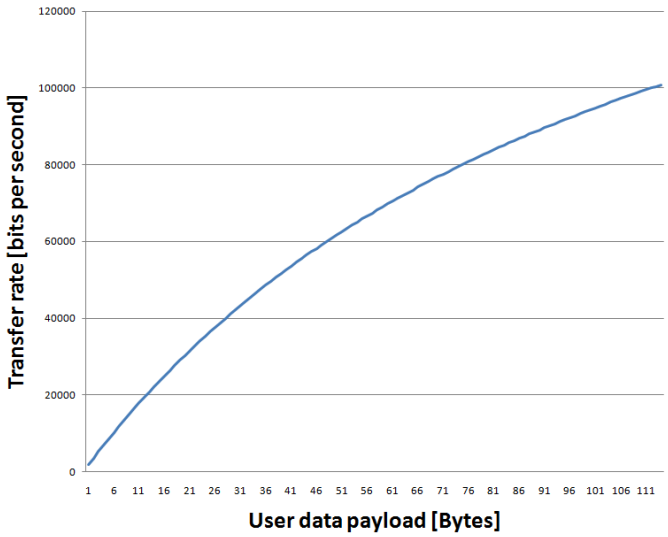


Figure 4.7: Calculated IEEE 802.15.4 transfer rates as a function of user data payload

4.5 A theoretical approach to the possible transfer rate of a star based motion capture system

IEEE 802.15.4 features like CSMA/CA and acknowledgements will improve the implemented star based motion capture system. These features will allow more sensor elements and the transfer rates will be increased. [9] describes how to calculate the effective transfer rate in a non-beacon enabled IEEE 802.15.4 transmission which uses an unslotted CSMA/CA mechanism (see Section 2.3.1).

Figure 4.7 is based on the calculations in [9]. This calculation can be seen in the appendix.

An IEEE 802.15.4 frame consisting data from one accelerometer (6 bytes) packet will have a transfer rate of 10170 bps. This means a 4.72 ms latency across the link. The receiver has to read multiple sensor elements. The following equation is a rough estimate of the time the receiver element uses for each reception:

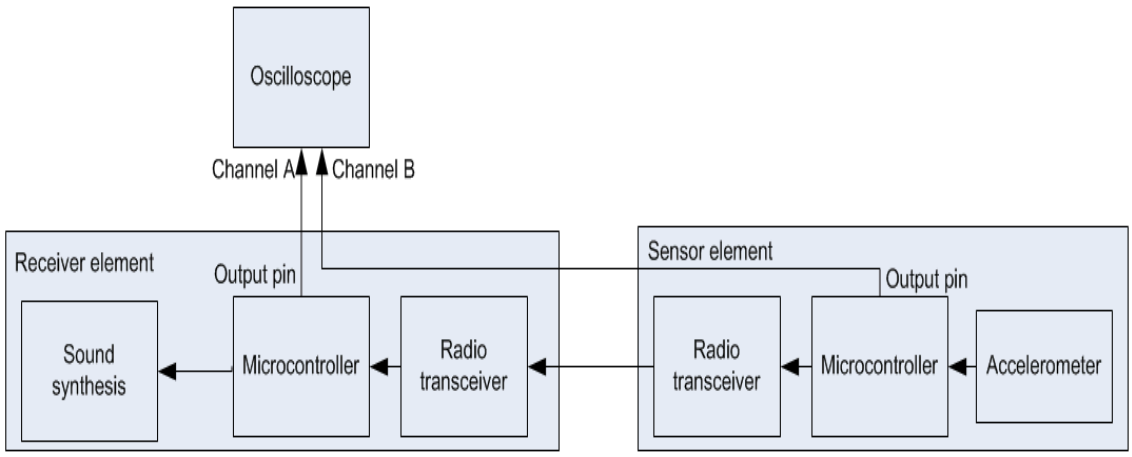


Figure 4.8: Drum setup

$$\begin{aligned} \text{IEEE 802.15.4 frame receiving time[s]} &= \text{Frame receiving[s]} \\ &+ \text{Turnaround Time[s]} \\ &+ \text{Acknowledgement transmission[s]} \\ &+ \text{Turnaround Time[s]} \end{aligned}$$

Turnaround time is the time needed for a device to switch between transmitter and receiver [9]. A realistic acknowledgement transmission time can be 0.352 ms [9].

$$\begin{aligned} \text{IEEE 802.15.4 frame receiving time[s]} &= 0.800ms \\ &+ 0.192ms \\ &+ 0.352ms \\ &+ 0.192ms \\ &= 1.54ms \end{aligned}$$

This can be used to calculate the possible number of sensor elements the receiver element can receive data from during 10 ms:

$$\text{Number of possible 6 byte transmissions during 10 ms} = \frac{10ms}{1.54ms} = 6.5 \qquad (4.4)$$

The receiver element can receive data from 6 sensor elements during 10 ms in a non-beacon enabled star based motion capture system with CSMA/CA .

4.6 An *Active Music* application

The implemented wireless motion based drum synthesis is described in 3.6. This implementation indicates the usability of the peer-to-peer based motion capture system in an *Active Music*

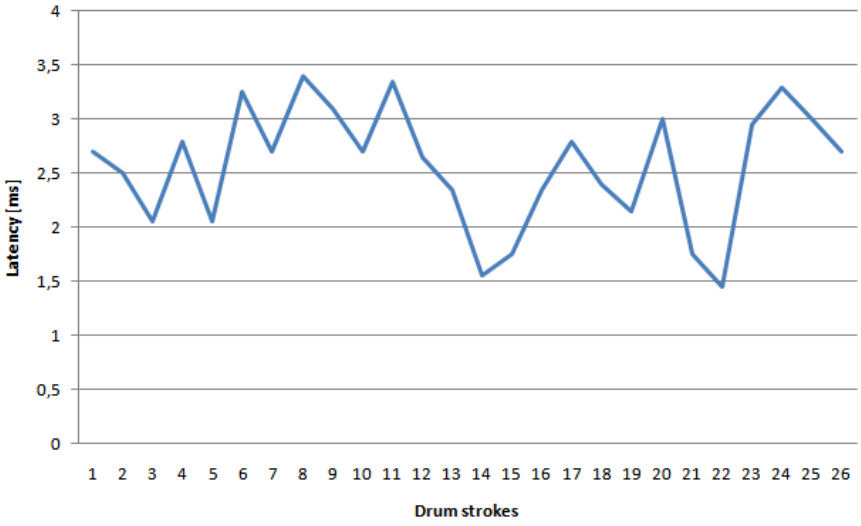


Figure 4.9: Latency variation across the wireless link

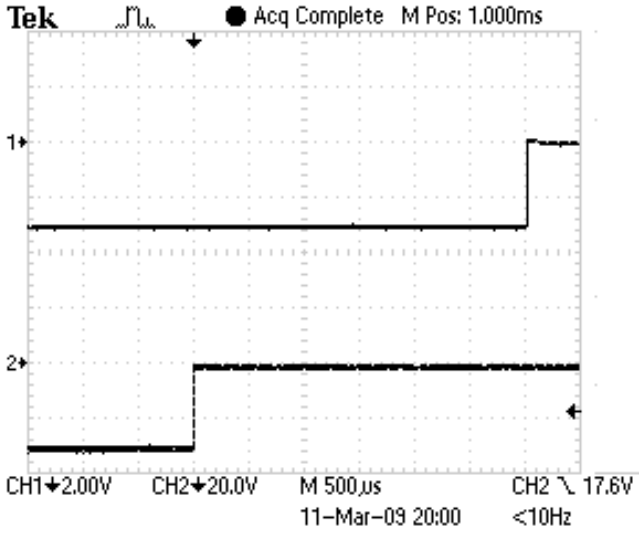


Figure 4.10: Latency measurement

application. The wireless drum-beat detector is a latency sensitive application, where latencies close to 10 ms will be noticeable.

This section describes a measurement of the latency across the wireless link. Figure 4.8 shows the measurement setup. The sensor element reads one accelerometer and transmits an IEEE 802.15.4 frame when the motions are above a given threshold. When this occurs, an output pin in the sensor element is set. The receiver element sets an output pin after it receives the IEEE 802.15.4 frame.

Figure 4.9 show the measured latency. The average is 2.5 ms. Figure 4.10 shows how the oscilloscope outputs the latency between the sensor element and the receiver element.

4.6.1 Conclusion

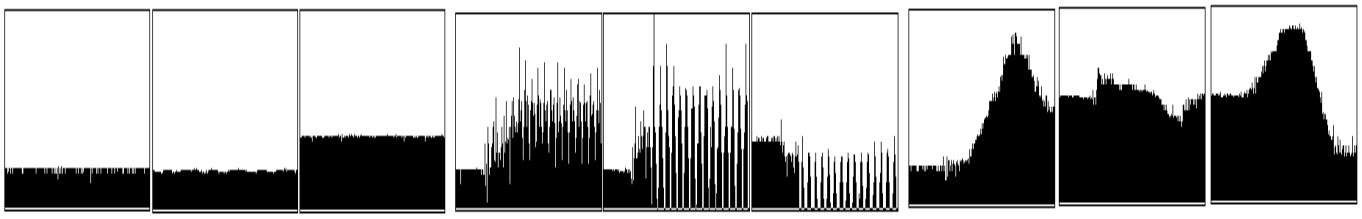
The measurements presented in this section shows the functionally of the peer-to-peer based motion capture system in an *Active Music* application. The measurements do not exceed the requirement of *Active Music* application. The ADC and the serial transmission latency are not included in this test. This setup uses one ADC channel which is considered insignificant[2]. The serial transmission latency is excluded because it will probably not be used in future *Active Music* applications.

4.7 Sensor data processing within the sensor element

This section will describe the amount of sensor data processing, which is reasonable to implement in the sensor elements. The results from the implemented processing algorithm described in Section 3.5 will be presented.

4.7.1 Filter implementations

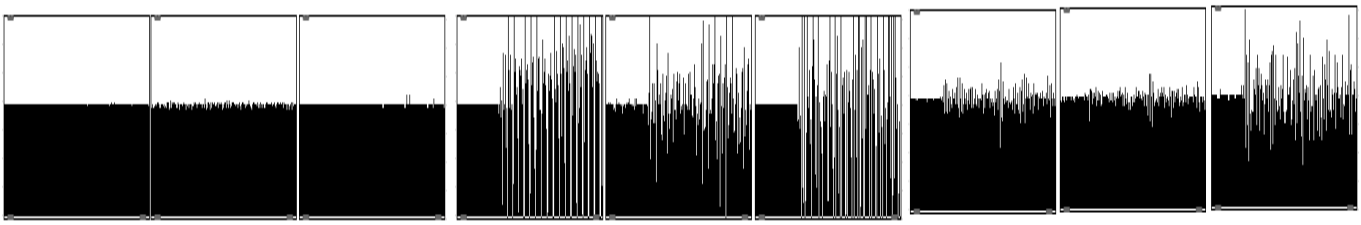
Section 3.5 describes an implementation of a median filter, a mean filter and a high pass filter. These filters were implemented in both the sensor element (in a microcontroller program) and the receiver element (in a MAX/MSP patch). Figure 4.11 shows un-processed accelerometer data. Figure 4.12 shows accelerometer data processed in the sensor element. Figure 4.13 shows accelerometer data processed in MAX/MSP. The MAX/MSP patches is shown in Section 3.5.



(a) No movement on the accelerometer (b) Fast shaking along the X-axis (c) Slow tilt

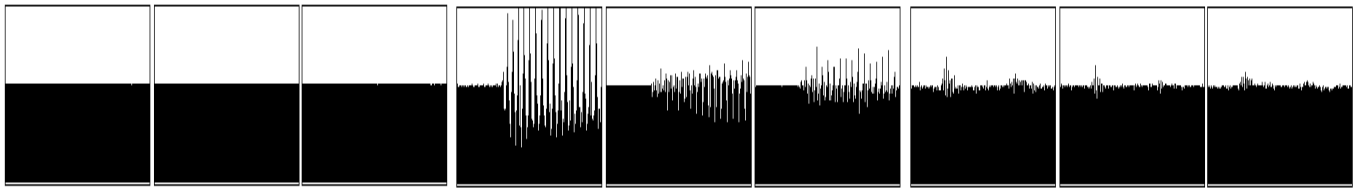
Figure 4.11: Without processing

[2] 13 μs ADC conversion time



(a) No movement on the accelerometer (b) Fast shaking along the X-axis (c) Slow tilt

Figure 4.12: Sensor data processing in the sensor element (microcontroller)



(a) No movement on the accelerometer (b) Fast shaking along the X-axis (c) Slow tilt

Figure 4.13: Sensor data processing in the receiver element (MAX/MSP)

Filters and algorithms are easier to implement in applications such as MAX/MSP than in a microcontroller. These graphs shows better results when processing data in MAX/MSP, because the MAX/MSP filters are more complex than the implemented sensor element filters. But as long as the microcontroller is waiting for a possibility to transmit an IEEE 802.15.4 frame, it can perform multiple ADC measurements and processing without gaining additional latencies. Other possibilities concerning receiver element processing is to fill an array with multiple ADC measurements, and transmit IEEE 802.15.4 frames with the ADC arrays to MAX/MSP for further processing.

Table 4.8 shows the time needed for median calculations. This includes the ADC measurements and the program execution time inside the microcontroller. This is based on a program counter feature in AVRstudio. These latencies are small compared to the ADC conversion latencies. Therefore, these latencies will not be added in the following calculations concerning sensor element processing time.

Median window size	f_{osc}	**ADC conversion time**	**processing time**	**Total time**
3	4 MHz	936 μs	295.50 μs	1.2315 ms
	8 MHz	936 μs	147.75 μs	1.08375 ms
	16 MHz	936 μs	73.875 μs	1.009875 ms
5	4 MHz	1.560 ms	749.50 μs	2.3095 ms
	8 MHz	1.560 ms	374.75 μs	1.93475 ms
	16 MHz	1.560 ms	187.375 μs	1.747375 ms

Table 4.8: The time needed for median calculations

	ADC conversion time	IEEE 802.15.4 latency	Possible processing time (10ms - IEEE 802.15.4 latency)	Possible number of ADC measurements for each accelerometer
1 accelerometer	312 μs	3.042 ms	6.958 ms	22
2 accelerometer	624 μs	3.203 ms	6.797 ms	10
3 accelerometer	936 μs	3.417 ms	6.583 ms	7
4 accelerometer	1.248 μs	3.628 ms	6.372 ms	5
5 accelerometer	1.560 μs	3.84 ms	6.16 ms	3

Table 4.9: The peer-to-peer based motion capture system implementation

	ADC conversion time	IEEE 802.15.4 latency	Possible processing time (10ms - IEEE 802.15.4 latency)	Possible number of ADC measurements for each sensor element
1 sensor element	312 μs	2.729 ms	7.271 ms	23
2 sensor elements	312 μs	5.232 ms	4.768 ms	15
3 sensor elements	312 μs	8.18 ms	1.82 ms	5
4 sensor elements	312 μs	10.03 ms	0 ms	0

Table 4.10: The star based motion capture system implementation

4.7.2 Conclusion

Several series of accelerometer data are needed for filtration and processing. The following graphs show the amount of ADC measurement which can be executed in the different implementations without exceeding 10 ms.

The peer-to-peer based motion capture system is a poor solution if multiple sensors have to be read and processed. This is shown in Table 4.9. A new series of data from 5 accelerometers will result in an additional 1.560 ms.

In a star based motion capture system, multiple sensor measurement can be conducted at the same time. Each sensor has its own microcontroller and the total ADC conversion time will be less than in the peer-to-peer network.

Table 4.11 shows how many ADC measurements a star based motion capture system can perform. The implemented star based motion capture system has limitation which results in a reduced possibility for sensor data processing. This is shown in Table 4.10.

	ADC conversion time	IEEE 802.15.4 latency	Possible processing time (10ms - IEEE 802.15.4 latency)	Possible number of ADC measurements for each sensor element
1 sensor element	312 μs	4.72 ms	5.28 ms	16
2 sensor elements	312 μs	4.72 ms	5.28 ms	16
3 sensor elements	312 μs	4.72 ms	5.28 ms	16
4 sensor elements	312 μs	4.72 ms	5.28 ms	16
5 sensor elements	312 μs	4.72 ms	5.28 ms	16
6 sensor elements	312 μs	4.72 ms	5.28 ms	16

Table 4.11: The estimated star based motion capture system

4.8 Discussion

The star network implemented in this thesis had major limitations. In order to get the data transmitted in the right order, the receiver change channel after each reception while the node was transmitting on different channels. The measurements in Section 4.4 shows a 50 percent reduction of the transfer rate for each sensor element when using two sensor elements instead of one.

Further addition of sensor elements will drastically decrease the transfer rate. The reason for this limitation was that AVR2002 transmission codes provided by Atmel didn't use the extended IEEE 802.15.4 features. These features handle collision avoidance in bigger IEEE 802.15.4 networks. An implementation of a CSMA/CA routine would have helped to avoid collision while increasing the transfer rate.

The peer-to-peer motion capture system implemented in this thesis was able to transfer data from multiple accelerometers within the latency requirements of 10 ms. This solution requires wires between the accelerometers and is therefore considered as not an optimal solution for a motion capture system.

The peer-to-peer setup used CSMA/CA and acknowledgements, while the star setup didn't. The reason for this was the fact that these implementations were based on existing codes for IEEE 802.15.4 transmission. These codes were hard to modify. Further adjustments will require much C-programming skills as well as a deep understanding of the timing requirements of the IEEE 802.15.4 standard.

The motion capture systems had some limitations that can be avoided. The serial transmission

inside the receiver element was a large limitation. The serial object in MAX/MSP wouldn't accept higher transfer rates than 76800. This became a problem when the receiver element had to transmit large amounts of data from several sensor elements into MAX/MSP. This latency can be ignored using for instance USB. A USB communication will have a minor latency component compared to other latency sources in the system.

The serial transmission inside the receiver elements influences the different setups in different ways. This transmission is most critical in the peer-to-peer based setup. When the receiver element receives a frame from the sensor element, the frame can consist of data from multiple accelerometers. The USART frames will therefore consist of a large amount of SLIP decoded accelerometer data. An additional 8.8 ms is applied if the receiver element serial transmites data from 5 accelerometers with 76400 bps USART.

In a star based motion capture system, the serial transmission will be executed after the reception of data from one sensor element. Therefore, each USART transmission will only include data from one accelerometer and this limitation will not be as critical as in the peer-to-peer based setup.

Chapter 5

Conclusion

Latencies caused by the serial transmission within the receiver elements have been ignored in this conclusion. Future improvements of this setup will use other more suitable standards for this communication.

A wireless peer-to-peer based motion capture system and a wireless star based motion capture system have been implemented in this thesis. These implementations have been tested for usability in *Active Music* applications. *Active Music* applications have latency requirements. The latency from motion to generated sound has to be below 10 ms. Latencies above this requirement will be noticeable for the performers.

The *Active Music* latency requirement influences the possible size of the implemented motion capture systems. These are the results considering the maximum number of sensor elements and accelerometers in the different setups:

- The implemented star based motion capture system can consist of 3 accelerometers. Each of these accelerometers is part of individual sensor elements. This is a complete wireless setup.
- An improved star based motion capture system can consist of 6 accelerometers. This is an estimate based on different calculations. Each of these 6 accelerometers will be part of individual sensor elements.
- The implemented peer-to-peer motion capture system has the possibility to read 5 accelerometers. This setup is limited by the microcontroller's available number of ADC channels. Also, this solution has limitations caused by the need for wires between the accelerometers and the microcontroller in the sensor element. Some applications will not be able to use this setup, for instance a setting where motions will be interfered by wires across the body.

This thesis also looked at the possibility for sensor data processing within the sensor elements. These results are based on how many ADC measurements there are possible to perform without exceeding the maximum time frame given by the *Active Music* latency requirement. The median filter is a used as an example to the amount of possible processing which can be carried out inside the sensor element. This filter removes noise and can be filled with a large amount of accelerometer data. This filter can be replaced with other suitable filters or algorithms. Here are the results considering the amount of processing which can be executed by the sensor elements in the implemented motion capture systems:

- The implemented star based motion capture system, which consists of 3 sensor elements, can measure 5 series of accelerometer data. This means that a median calculation which includes 5 series of accelerometer data can be conducted.
- The improved star based motion capture system, which consists of 6 sensor elements, can measure 16 series of accelerometer data. This means that a median calculation which includes 16 series of accelerometer data can be conducted.
- The implemented peer-to-peer based motion capture system, which consists of 5 accelerometers, can measure 3 series of accelerometer data. This means that a median calculation which includes 3 series of accelerometer data can be conducted.

The star based motion capture system is considered most suitable for *Active Music* of the implemented motion capture systems. The star based motion capture system has advantages considering latencies and sensor data processing inside the sensor elements. Each sensor element reads and process sensor data from one accelerometer. This is advantageous since processing on different accelerometers will be executed in parallel. The limitations concerning channel change and collision avoidance can be overcome by improvements described in future works (Chapter 6). This improvement will increase the possible amount of sensor elements.

Chapter 6

Future works

There are many possible improvements which can increase the performance of the implementations done in this thesis:

- The extended IEEE 802.15.4 features like CSMA/CA and acknowledgments have to be implemented in order to get an effective star based motion capture setup. Also, a more effective communication within the receiver elements has to be implemented. This can be a high speed standard such as USB. The RZUSBSTICK, used in this thesis, consist of a USB microcontroller which can be programmed to exploit the USB standard.
- The program flows implemented in this thesis can be improved considering sensor element processing. My implementations await an IEEE 802.15.4 frame transmissions until all sensor measurements and processing is conducted. A parallel program flow can increase the time for additional processing in the sensor elements. A new series of ADC measurements could be converted at the same time as the former ADC measurements are transmitted to the receiver element.
- Beacon enabled IEEE 802.15.4 networks (see Section 2.3.1) have not been looked at in this thesis. This can be a possible improvement of the entire system.
- A full Zigbee implementation will provide possible topologies like mesh and tree. This can be exploited in a future improvement if there is a need for more complex topologies.
- The peer-to-peer based setup can be turned into a multiple peer-to-peer network where each sensor element has one receiver element. In this setup the sensor elements don't have to measure multiple accelerometers, which is a limitation in the implemented peer-to-peer based motion capture system.

Bibliography

[1] Analog devices, www.analog.com/static/imported-files/data_sheets/ADXL330.pdf. *Analog devices ADXL330 accelerometer datasheet*, 2007.

[2] Atmel, www.atmel.com/dyn/resources/prod_documents/doc2491.pdf. *Atmel STK501 user guide*, September 2001.

[3] Atmel, http://www.atmel.com/dyn/resources/prod_documents/doc1925.pdf. *Atmel STK500 user guide*, Mars 2003.

[4] Atmel, www.atmel.com/dyn/resources/prod_documents/doc5182.pdf. *Atmel IEEE 802.15.4 MAC user guide*, September 2006.

[5] Atmel, www.atmel.com/dyn/resources/prod_documents/doc8087.pdf. *Atmel AT86RF230 Software Programmer's Guide*, June 2007.

[6] Atmel, http://www.atmel.com/dyn/resources/prod_documents/doc2549.pdf. *Atmel ATMega1281v microcontroller datasheet*, August 2007.

[7] Atmel, http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf. *Atmel at86rf230 radio transceiver datasheet*, February 2009.

[8] Sinem Coleri Ergen. *Zigbee/IEEE 802.15.4 Summary*. berkeley, 2004.

[9] Jennic, www.jennic.com/download_file.php?supportFile=JN-AN-1035%20Calculating%20802-15-4%20Data%20Rates-1v0.pdf. *Calculating 802.15.4 Data Rates*, 2006.

[10] Patrick Kinney. Zigbee technology: Wireless control that simply works. www.Zigbee.org/resources, October 2003. Whitepaper.

[11] John Kooker. Bluetooth, zigbee, and wibree: A comparison of wpan technologies. http://www.cse.ucsd.edu/classes/fa08/cse237a/topicresearch/jkooker_tr_report.pdf, November 2008.

[12] J.-S. Lee. Performance evaluation of ieee 802.15.4 for low-rate wireless personal area networks. *Consumer Electronics, IEEE Transactions on*, 52(3):742–749, Aug. 2006.

[13] Joseph W. Malloch. *A Consort of Gestural Musical Controllers: Design, Construction, and Performance*. McGill University, 2008.

[14] Curtis Roads. *The Computer Music Tutorial*. MIT Press, Cambridge, MA, USA, 1995.

[15] Jim Tørresen Rolf Inge Godøy, Mats Høvin and Alexander Jensenius. *Sensing Music related Actions*. University of Oslo, 2007.

[16] Arve Voldsund. "Mapping av sensordata til DSP-funksjoner". Master's thesis, Department of Musicology, University in Oslo, Norway, 2007.

[17] David Wessel and Matthew Wright. Problems and prospects for intimate musical control of computers. In *NIME '01: Proceedings of the 2001 conference on New interfaces for musical expression*, pages 1–4, Singapore, Singapore, 2001. National University of Singapore.

[18] Wikibooks. Embedded systems/atmel avr. `http://en.wikibooks.org/wiki/Embedded_Systems/Atmel_AVR`.

[19] Wikipedia. Csma-ca. `http://en.wikipedia.org/wiki/Carrier_sense_multiple_access_with_collision_avoidance`.

[20] Wikipedia. The osi model. `http://en.wikipedia.org/wiki/OSI_model`.

[21] Wikipedia. The slip protocol. `http://en.wikipedia.org/wiki/SLIP`.

Appendix A

CD contents

The CD contains the following:

- The adjusted AVR2001 application note for accelerometer measurements.
- The adjusted AVR2001 application note for transfer rate measurements.
- The adjusted AVR2001 application note for the drum application.
- The adjusted AVR2002 application note for accelerometer measurements.
- The adjusted AVR2002 application note for transfer rate measurements.
- MAX/MSP patches for sensor data monitoring and sensor data processing.

Appendix B

Sensor element processing algorithm

```
1
#include <avr/io.h>          // Load AVR definitions
3 #include <avr/signal.h>     // Load interrupt handling
#include <avr/interrupt.h> // Load interrupt flag control
5 #include "math.h"
#include "stdio.h"
7 #define FOSC 8000000// Clock Speed
#define axis 3
9 #define N_samples 9 // antall målinger på hver akse/ved median må være
    oddetall, avarage = feks 100
#define ADC_channels 3 //antall akser
11 #define ENABLE_RECEIVER   ( UCSR0B |= ( 1 << RXEN0 ) ) /*!< Enables receiver
    . */
#define DISABLE_RECEIVER ( UCSR0B &= ~( 1 << RXEN0 ) ) /*!< Disables
    receiver. */
13 #define ENABLE_TRANSMITTER   ( UCSR0B |= ( 1 << TXEN0 ) ) /*!< Enables
    transmitter. */
#define DISABLE_TRANSMITTER ( UCSR0B &= ~( 1 << TXEN0 ) ) /*!< Disables
    transmitter. */
15 #define ENABLE_RECEIVE_COMPLETE_INTERRUPT   ( UCSR0B |= ( 1 << RXCIE0 ) )
    /*!< Enables an interrupt each time the receiver completes a symbol. */
#define DISABLE_RECEIVE_COMPLETE_INTERRUPT ( UCSR0B &= ~( 1 << RXCIE0 ) )
    /*!< Disables an interrupt each time the receiver completes a symbol. */
17

19 volatile unsigned int temp1[3], temp2[3], temp3[3], median_count,
    mean_count, median_table[axis][N_samples], mean_table[axis][N_samples];

21 volatile unsigned int temp_table[axis]={600,600,600}, send_value; //lagre
volatile unsigned int mux=0, send_table[3], send_table2[3], r, t1, t2, t3;
23 volatile unsigned int flag, median[], t, avarage[], send[], send_pres[];
volatile unsigned int send_high[axis], send_int,velocity;
25 volatile unsigned int send_low[axis], ADC_send[3],ADC_sendH[axis],
    ADC_sendL[axis], sendL[3], sendH[3], tempH[3], temp[3];
volatile unsigned int ADC_high[ADC_channels], ADC_table[axis][N_samples];
27 volatile unsigned int ADC_low[ADC_channels];

29 // Main function
int main (void) {
31
```

```
    //Initialize USART module.
33  UBRR0H = 0x00;
    UBRR0L = 0x0c;
35  //Enable USART transmitter module. Always on.
    ENABLE_RECEIVER;
37  ENABLE_TRANSMITTER;
    //8-N-1.
39  UCSR0C |= ( 1 << UCSZ01 ) | ( 1 << UCSZ00 );
    ADMUX = (0<<REFS1) | (1<<REFS0) | (0<<ADLAR) | (0<<MUX0);
41  ADCSRA |= (0 << ADPS2) | (0 << ADPS1) | (1 << ADPS0);
    ADCSRA |= (0 << ADATE) | (1 << ADEN) | (1 << ADIE);
43  ADCSRA |=(1 << ADSC);

45  sei();

47
    while(1){
49  if(flag==1){

51
    if(mean_count==N_samples){
53

55  //Avarage filter
        for(int k=0; k<mean_count; k++)//adds numbers in each row
57        {
          avarage[0]+=mean_table[0][k];//+avarage[0];
59        avarage[1]+=mean_table[1][k];//+avarage[1];
          avarage[2]+=mean_table[2][k];//+avarage[2];
61

63        }

65
      send_pres[2]=avarage[2]/N_samples;
67    send_pres[0]=avarage[0]/N_samples;
      send_pres[1]=avarage[1]/N_samples;
69

71  //High pass filter
      for(int h = 0;  h < axis; h++){
73      temp1[h]=send_pres[h]-temp_table[h];
        temp2[h]=10*temp1[h];
75      send[h]=temp2[h]+512;
        temp_table[h]=send_pres[h];
77    }

79

81  for(int j=0;j<axis;j++){

83        sendL[j]=(send[j] & 0xFF);
        sendH[j]=send[j]>>8;
85
```

```
87
        }
89 //SLIP encoded USART transmission of the processed data

91      for(int i=0;i<axis;i++){
          for(; !(UCSR0A & (1 << UDRE0));) {;}
93            UDR0 = 0300; //Put symbol in data register.
          for(; !(UCSR0A & (1 << UDRE0));) {;}
95          UDR0 = sendH[i];//temp1;
          for(; !(UCSR0A & (1 << UDRE0));) {;}
97            UDR0 = 0300; //Put symbol in data register.
          for(; !(UCSR0A & (1 << UDRE0));) {;}
99          UDR0 = sendL[i];//temp1;
        }
101       for(; !(UCSR0A & (1 << UDRE0));) {;}
            UDR0 = 0301; //Put symbol in data register.
103

105     for(int i=0; i<axis; i++)
          {
107       avarage[i]=0;
          temp[i]=0;
109       send_table[i]=0;
          sendL[i]=0;
111       sendH[i]=0;
          send[i]=0;
113       send_pres[i]=0;
          }
115
        for(int i=0; i<N_samples; i++)
117       {
          median_table[0][i]=0;
119       mean_table[0][i]=0;
          median_table[1][i]=0;
121       mean_table[1][i]=0;
          median_table[2][i]=0;
123       mean_table[2][i]=0;
        }
125   mean_count=0;
        ADCSRA |=(1 << ADSC);
127   }//end mean_count==N_samples

129   else {

131 //median calculation
      if(median_count==N_samples){
133     for(int i=0; i<=N_samples-1; i++){
          for(int j=i+1; j<N_samples;j++){
135         for(int l=0; l<axis; l++){
              if(median_table[l][i]>median_table[l][j])
137             {
                temp[l] = median_table[l][i];
139             median_table[l][i] = median_table[l][j];
                median_table[l][j] = temp[l];
141             }
```

```
                }
143           }
            }
145
          t=N_samples/2;
147       for(int k=0;k<axis;k++){
            median[k]=median_table[k][t];
149         send_table2[k]=median[k];
            }
151
          //fyll opp mean_table
153
          for(int i=0; i<axis;i++){
155       mean_table[i][mean_count]=send_table2[i];
          }
157       mean_count++;
          median_count=0;
159       ADCSRA |=(1 << ADSC);
          }//end median_count==N_samlples
161
        else{
163
          for(int i=0; i<axis;i++){
165       median_table[i][median_count]=ADC_send[i];
          }
167       median_count++;
          ADCSRA |=(1 << ADSC);
169     }//end else}

171
ADCSRA |=(1 << ADSC);
173     flag=0;
      }//end else
175   }//end flag==1
}//end while
177
}//end main
179

181 #pragma vector=ADC_vect
ISR(ADC_vect)
183   {

185 ADC_send[mux]=ADC;

187     if(mux<axis-1){
          mux++;
189     ADMUX=(ADMUX & 0xF8) | mux;
          ADCSRA |=(1 << ADSC);
191     }
      else{
193
          mux=0;
195     ADMUX=(ADMUX & 0xF8) | mux;
        ADCSRA |=(0 << ADSC);
```

```
197
            flag=1;
199         }
    }
```

Appendix C

CSMA/CA routines

After building IEEE 802.15.4 frames to be sent, the microcontroller put the radio into TX_ARET_ON state. This is one of the extended states, where the frame transmission uses an Automatic CSMA/CA retry. By looking at the frame field in the received TX_FRAME, which is sent from the microcontroller, the radio transceiver knows if an Acknowledgment is expected. If an Acknowledgment is used, the radio transceiver switches into RX_STATE after transmitting, and awaits the Acknowledgment frame. If no Acknowledgement frame is received within 864 μs, the radio transceiver retries the transmission. The number of retries can be set by the user. The radio transceiver retries the transmission until an Acknowledgment is received or the maximum number of retries is reached.

After building IEEE 802.15.4 frames to be sent, the microcontroller put the radio into PLL_ON state. This is one of the basic states, where the frames are transmitted without any Automatic CSMA/CA retry.

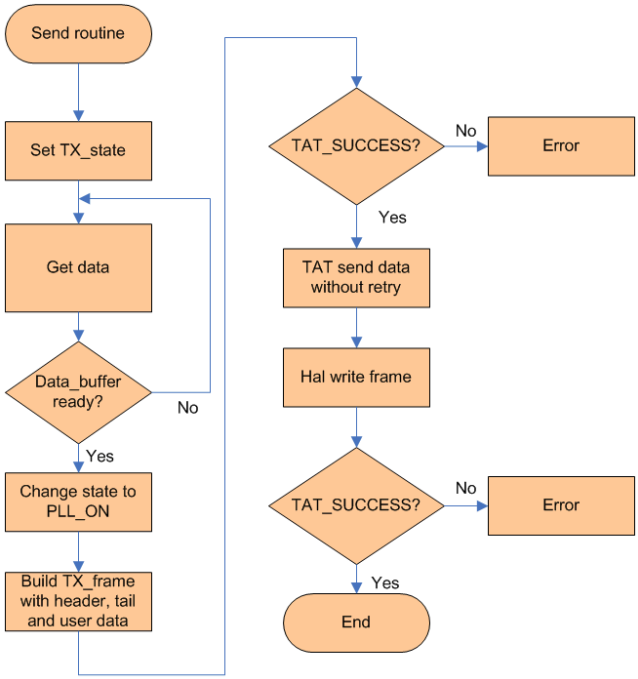


Figure C.1: IEEE 802.15.4 frame transmission routine

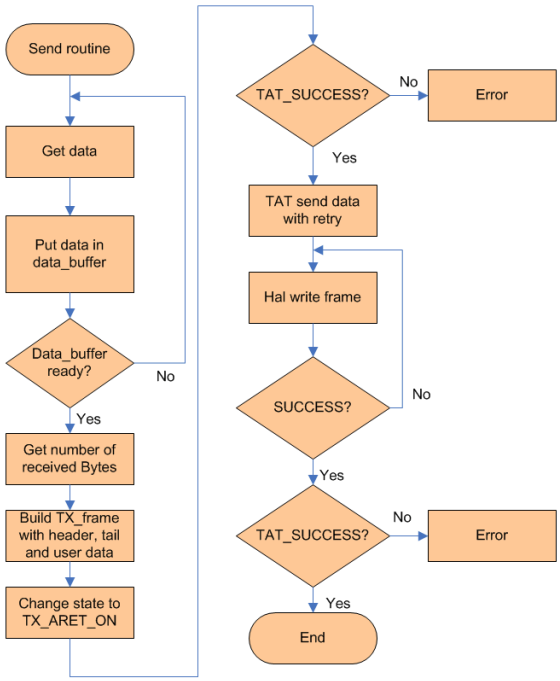


Figure C.2: IEEE 802.15.4 frame transmission routine

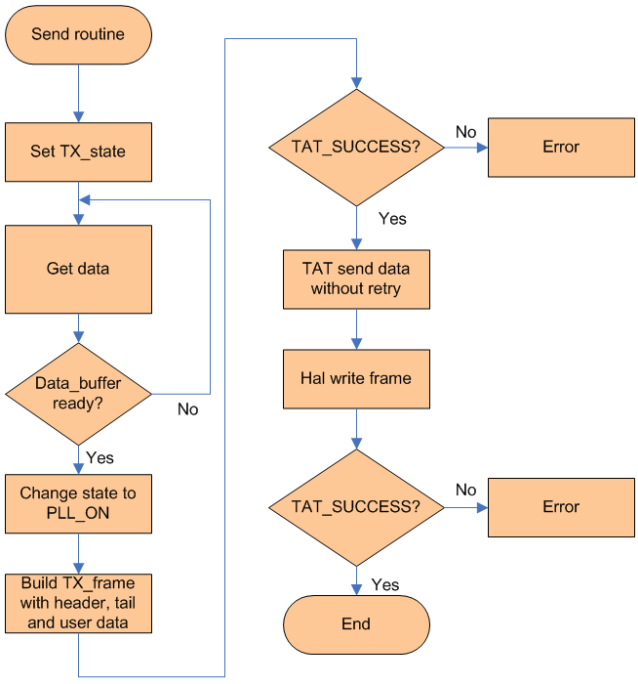


Figure C.3: IEEE 802.15.4 frame transmission routine

Appendix D

Jennic application note

Jennic
TECHNOLOGY FOR A CHANGING WORLD

Application Note: JN-AN-1035

Calculating 802.15.4 Data Rates

This Application Note describes how to calculate the effective and actual data rates of an IEEE 802.15.4 wireless network. The 802.15.4 data frame structure and channel access protocol overhead are described, and an example calculation for transferring 1 MB of data is given.

Application Overview

The IEEE 802.15.4 standard for Low-Rate Wireless Personal Area Networks (LR-WPANs) supports a maximum over-the-air data rate of 250 kbps for the 2400-MHz band. In practice, the effective data rate is somewhat lower due to the protocol built into the frame structure. Various mechanisms are also employed to ensure robust data transmission. These include channel access algorithms, data verification and frame acknowledgement. The data frame structure and associated protocol overhead are described, and used to determine the maximum data payload and packet transmission time. The channel access and frame acknowledge times are also calculated. An example actual data rate calculation is given for a non-beacon enabled network using unslotted CSMA-CA (with acknowledgements) and a transmission time calculated for 1 MB of data.

Note: The effective handling of the 802.15.4 protocol is critical in order to maintain the maximum data rate of the system. Depending on the system architecture, the processor overhead imposed by the user application program can often compromise data rates. The JN5121 incorporates a hardware implemented Base Band Controller. This handles the PHY access protocol such as Clear Channel Assessment, auto-acknowledge, packet retries and CRC Checking *without* the need for processor intervention. The JN5121 also provides an Intelligent Peripheral SPI interface (up to 8 Mbps), and also supports transfer rates of up to 1 Mbps on the UART.

Channel Access Timing

Non-beacon enabled IEEE 802.15.4 networks use an unslotted CSMA-CA channel access mechanism. This algorithm is shown in the Appendix, Figure 5. Each time a device needs to transmit, it waits for a random number of unit back-off periods in the range {0, $2^{BE} - 1$} before performing the Clear Channel Assessment (CCA).

- If the channel is found to be idle, the device transmits.
- If the channel is found to be busy, the device waits another random period before trying to access the channel again.

Initially, the back-off exponent *BE* is set to *macMinBE*. Using the default value of 3 for *macMinBE* and assuming the channel is found to be free, the worst-case channel access time can be calculated as:

InitialbackoffPeriod + *CCA* = (2^3 – 1) x *aUnitBackoffPeriod* + *CCA*

= 7 x 320 µs + 128 µs

= 2.368 ms

The CCA detection time is defined as 8 symbol periods.
aUnitBackoffPeriod is defined as 20 symbol periods.
1 symbol period is equal to 16 µs.

Note: If the *macMinBE* value is set to 0, collision avoidance is disabled during the first iteration of the algorithm. In this case, the channel access timing is defined by the minimum Inter-Frame Separation (IFS) constants *aMinSIFSPeriod* and *aMinLIFSPeriod* [1].

Maximum Data Payload

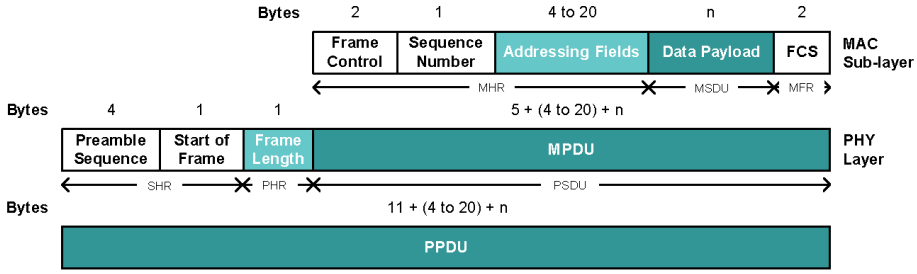


Figure 1: Schematic View of Data Frame

The IEEE 802.15.4 standard [1] specifies the maximum number of bytes that can be transmitted in the MAC data payload as 102 bytes:

aMaxMACFrameSize (MSDU) = aMaxPHYPacketSize (MPDU) – aMaxFrameOverhead

aMaxFrameOverhead = 25
aMaxPHYPacketSize = 127

The IEEE 802.15.4 standard defines *aMaxMACFrameSize* using the maximum frame overhead regardless of the actual frame overhead size. The more recent IEEE 802.15.4b standard allows the maximum data payload to be larger when fewer addressing fields are being used. The JN5121 supports this 802.15.4b feature - using short addressing (16-bit source and destination addresses), the frame overhead is reduced to 13 bytes, leaving a data payload of **114 bytes** (see Figure 2).

Bytes: 2	1	0/2	0/2/8	0/2	0/2/8	variable	2
Frame control	Sequence number	Destination PAN ID	Destination address	Source PAN ID	Source address	Frame payload	FCS
		Addressing fields					
MHR						MAC payload	MFR

Figure 2: MAC Protocol Data Unit (MPDU)

Note: For the Star network topology, it is possible to specify only source addressing fields, increasing the maximum data payload to **118 bytes**. In this situation, the frame is accepted only if the device is a PAN Coordinator and the source PAN identifier matches that of the PAN Coordinator.

Data Frame Transfer Time

Adding the 6-byte packet overhead (Preamble and Start of Frame Delimiter [SHR], and Frame Length [PHR]) to the MAC Protocol Data Unit (MPDU), and given a fundamental data rate into the modem of 250 kbps, the frame transfer time is calculated as:

$$\frac{(aMaxPHYPacketSize + SHR + PHR) \times 8}{250 \times 10^3} = \frac{(127 + 5 + 1) \times 8}{250 \times 10^3} = 4.256ms$$

(for a maximum data payload of 114 bytes)

Acknowledged Transmission Timing

An acknowledgment frame consists of 11 bytes and is shown in Figure 3. Given a fundamental data rate into the modem of 250 kbps, transmission takes **0.352 ms**. The transmission of an acknowledgement does not use CSMA/CA.

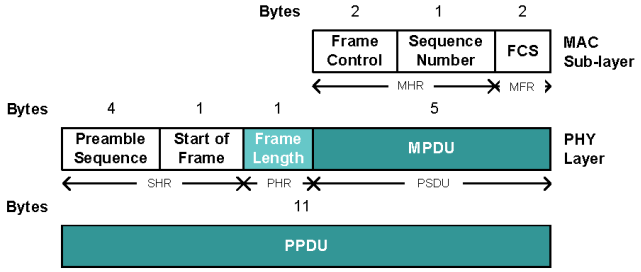


Figure 3: Schematic View of Acknowledgement Frame

The transmission of an acknowledgment frame (in a non-beacon enabled network) commences *aTurnaroundTime* symbols after the reception of the data frame, where *aTurnaroundTime* is equal to **192 µs**. This allows the device enough time to switch between transmit and receive, or vice versa. The timing for an acknowledged transmission is shown in Figure 4 below.

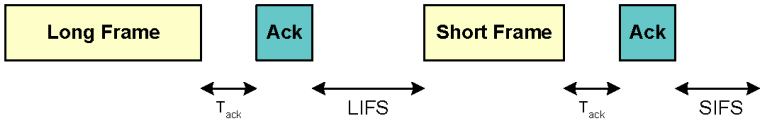


Figure 4: Acknowledged Transmission Timing

(LIFS = Long Inter-Frame Spacing, SIFS = Short Inter-Frame Spacing)

To allow the received data to be processed, the acknowledgement frame is followed by a minimum inter frame separation period (IFS). The length of the IFS period is dependent on the frame size. Frames (MPDUs) of up to 18 bytes in length must be followed by a SIFS period of at least 12 symbol periods. Frames with lengths greater than 18 bytes must be followed by a LIFS of at least 40 symbol periods. This defined minimum delay is usually absorbed by the CSMA-CA channel access timing.

Retry Timing

In practice, it is unlikely that a 0% PER will be achieved. The transmitting node will wait *macAckWaitDuration* symbol periods for an acknowledgment before it attempts a retry, where *macAckWaitDuration* is equal to 54 symbol periods (0.864 ms).

Effective Data Rate

From the information detailed above, an effective data rate can be calculated based on the following assumptions:

- Non-beacon enabled network
- CSMA/CA algorithm never finds that the channel is busy
- Includes subsequent reception of associated acknowledgment
- No retries are required
- Maximum data payload is 114 bytes

The calculation is as follows:

CSMA/CA (data frame)	2.368 ms (default random back-off exponent of 3)
Data frame transmission	4.256 ms
Turnaround time	0.192 ms
Acknowledgement transmission	0.352 ms
Total	7.168 ms
Effective data rate	(114 x 8) / (7.168 x 10^{-3}) = **127 kbps**

Actual Data Rate

An estimate of the actual data rate can be calculated for a non-ideal environment by allowing for retries. Assuming a packet error rate of 25% and all packets that require a retry only require one retry, the total time to transmit a frame with one retry is:

CSMA/CA (data frame)	2.368 ms (default random back-off exponent of 3)
Data frame transmission	4.256 ms
MacAckWaitDuration	0.864 ms
CSMA/CA (data frame)	2.368 ms
Data frame transmission	4.256 ms
Turnaround time	0.192 ms
Acknowledgement transmission	0.352 ms
Total	14.656 ms

The time to transmit a frame with no retries is as calculated for the effective data rate: 7.168 ms.

Therefore, with 75% of data frames taking 7.168 ms and 25% of data frames taking 14.656 ms:

Average data frame transmission time = (7.168 x 0.75) + (14.656 x 0.25) = 9.04 ms

Assuming a 114-byte payload, the achievable data rate is then given by:

Actual data rate = (114 x 8) / (9.04 x 10^{-3}) = **101 kbps**

So, the transfer time for 1 MB of data will be:

$$\frac{2^{20}}{114} \times 9.04ms = 1\,\mathbf{min}\ 23\,\mathbf{sec}$$

To improve on this, we could use a random back-off exponent of 1 rather than the default value of 3. In this case, the CSMA/CA channel access time will default to the minimum Long Inter-Frame Spacing (LIFS) of 0.640 ms. This gives an actual data rate of 133 kbps and a transfer time of ~1 minute.